UNIVERSIDADE DE LISBOA
FACULDADE DE CIÊNCIAS
DEPARTAMENTO DE INFORMÁTICA

UNIFIED CYBER THREAT INTELLIGENCE

Marisa Tomé Félix

MESTRADO EM INFORMÁTICA

Dissertação orientada por:
Prof. Doutor António Casimiro Ferreira da Costa
e pelo Eng. José António do Santos Alegria

2018

Agradecimentos

Em primeiro lugar, gostaria de agradecer ao Professor Doutor António Casimiro, pela sua disponibilidade, apoio e orientação ao longo de todo este projeto. Gostaria de agradecer ao Engenheiro José Alegria, o orientador na Portugal Telecom, pela oportunidade que me foi oferecida de poder trabalhar e aprender na sua equipa, assim como por todos os conselhos e conhecimentos que me transmitiu ao longo de todo o estágio.

Ao meu orientador técnico e mentor Pedro Reis, por toda a paciência que teve comigo, por tudo o que me ensinou e pelo aconselhamento que me foi dado para a conclusão deste projeto. Também gostaria de agradecer ao Pedro Gonçalves e ao Pedro Inácio, pois contribuíram para o meu ingresso neste estágio, dando-me uma oportunidade de aprendizagem e desenvolvimento, tanto a nível profissional como pessoal.

Quero agradecer a toda a equipa da DCY, por me terem recebido como se já fizesse parte da equipa desde sempre.

Devo um agradecimento especial ao meu colega de estágio Pedro Santos, pela amizade, aconselhamento e pela ajuda imprescindível ao longo de todo o estágio.

Também gostaria de agradecer à minha colega e amiga Patrícia Abrantes, pela amizade e suporte durante todo o meu ingresso no Mestrado em Informática.

Um agradecimento especial para minha melhor amiga Rita Aires, por ter estado sempre ao meu lado, pelos seus conselhos e apoio e por todos estes anos de amizade.

Quero agradecer ao meu namorado Ricardo Lima, pela paciência que tem comigo, pela ajuda que me deu durante este projeto, e por estar sempre ao meu lado quando mais preciso.

Finalmente, gostaria de agradecer à minha família, aos meus pais e irmão, por me terem dado o suporte necessário e ferramentas para o alcance e conclusão de todos os meus objetivos.

Aos meus Pais.

Resumo

Ao longo dos anos, a preocupação com a Ciber Segurança (a proteção de sistemas, redes e de informações num ciber espaço) nas grandes empresas tem vindo a aumentar, isto porque, atualmente a maioria das organizações depende de dados informatizados e partilham grandes quantidades de informação por todo o globo, tornando-se em alvos mais fáceis para muitas formas de ataque. Consequentemente, um ciberataque pode prejudicar o nome e a reputação de uma empresa, resultando na perda de vantagem competitiva, criando um incumprimento legal / regulamentar e causando danos financeiros.

De modo a evitar um possível comprometimento nas infraestruturas de uma organização, é necessário tomar medidas de precaução, isto é, fazer uma análise e gestão dos riscos a que uma empresa está exposta e assim delinear uma estratégia, de maneira a minimizar, mitigar e / ou anticipar ataques.

A Portugal Telecom, conhecida também por PT Portugal ou Grupo PT, tratando-se da maior operadora de telecomunicações em Portugal, não descura da preocupação com a Ciber Segurança. Como tal, esta possui uma direção dedicada à segurança e privacidade da informação, a Direção de Cyber Security and Privacy (DCY).

Para que haja uma proteção ciber resiliente nas infraestruturas / ativos da PT, a DCY divide-se em diferentes operações:

- *Cyber Security Governance*: responsável pela gestão das operações / programas de Ciber Segurança da DCY, incutindo objetivos a cada uma das operações;
- *Cyber Security Operations*: responsável pela resposta a incidentes;
- *Cyber Watch*: responsável pela análise proativa de riscos, isto é, identificação dos vários ativos de informação que podem ser afetados por um ciberataque e monitorizar continuamente o ambiente de risco.

O projeto *"Unified Cyber Threat Intelligence"* encontra-se dentro da operação de Cyber Watch da DCY e está a ser desenvolvido na Portugal Telecom, no âmbito da disciplina PEI (Projeto em Engenharia Informática) do Mestrado em Informática (MI) da Faculdade de Ciências da Universidade de Lisboa (FCUL).

A Cyber Watch é composta por diferentes áreas, como por exemplo, *Cyber Higiene*, *Cyber Awareness* ou *Cyber Intelligence*, de maneira a que a análise proativa de riscos seja

o mais eficaz possível. O foco deste projeto dentro da Cyber Watch insere-se dentro da *Cyber Intelligence*.

A *Cyber Intelligence* é um tipo de informação que fornece a uma organização suporte nas decisões, levando a uma vantagem estratégica proporcionando aos utilizadores informações constantemente atualizadas sobre possíveis fontes de ataque.

No contexto da Portugal Telecom, a *Cyber Intelligence* decompõe-se em diferentes ramificações, sendo as mais importantes para este projeto as fontes de risco (*risky sources*), estas referem-se a todos eventos ocorridos na Internet no geral e os alvos comprometidos (*compromised targets*), tratando-se dos eventos que ocorreram dentro da organização.

O objetivo principal deste projeto é a implementação de uma arquitetura escalável para a recolha, análise, remoção de duplicados, classificação, etiquetagem e filtragem de *Cyber Intelligence Events*, sendo estes aplicados no contexto de organizações, unidades de negócio, infraestruturas, ativos ou atores. Desta forma, será possível realizar uma análise forense aos *Cyber Intelligence Events* recolhidos, de modo a que haja uma melhor compreensão sobre o tipo de ataques a que as organizações estão expostas.

Um evento, no contexto da Cyber Watch da Portugal Telecom, trata-se de um facto ou ocorrência observável na Internet pública (envolvendo endereços IP e / ou Fully Qualified Domain Names (FQDNs)), onde este aconteceu num certo período de tempo. Para o processamento deste tipo de eventos utilizar-se-á o IntelMQ. Este trata-se de uma plataforma para recolher e processar *feeds* de segurança, isto é, correntes de informação composta por factos e evidências de que um certo evento aconteceu, como por exemplo, endereços IP ou domínios que estão envolvidos em atividades maliciosas.

O IntelMQ tem como objetivo principal ajudar analistas de ciber segurança a recolher e processar *Cyber Intelligence Events* permitindo o redirecionamento / envio da informação tratada para outros sistemas. Para que um evento IntelMQ seja válido são necessários certos requisitos para que este faça sentido. Para tal, os requisitos mínimos são:

- O **nome** da *Feed* de Segurança onde o IntelMQ foi coletar o evento;
- O **tipo** de evento que foi encontrado, por exemplo, *spam* ou *malware*;
- A **taxonomia** do evento, por exemplo, Conteúdo Abusivo (poderá estar associado a *spam*) ou Código Malicioso (poderá estar associado a *malware*);
- O **tempo de origem**, a hora e data reportados por uma fonte de informação (*feed*);
- O **tempo de observação**, a hora e data em que o IntelMQ processou o evento.

Adicionalmente, um evento IntelMQ deverá conter pelo menos um dos seguintes campos:

- **IP de origem**, o endereço IP observado que iniciou uma ligação;
- **FQDN de origem**, o nome DNS relacionado a um *host* de onde originou a ligação;

- **URL de origem**, refere-se a um recurso mal-intencionado onde a sua interpretação é definida pelo tipo de abuso, por exemplo, um URL em que o abuso seja do tipo *phishing* refere-se a um recurso de *phishing*;

- **Conta de origem**, nome de uma conta ou um endereço de e-mail relacionado com a origem de um evento.

A recolha e filtragem de *Cyber Intelligence Events* será direcionada a entidades-alvo e será obtida através de diferentes fontes de informação, como por exemplo, *open-source / paid intelligence feeds*. Para este projeto foram utilizadas mais de trinta fontes de informação. Segue-se um pequeno exemplo de fontes já existentes no IntelMQ:

- Abuse.ch Ransomware Tracker: fornece listas de FQDNs, URLs e endereços IP que foram utilizados por diversas famílias de *ransomware*;

- PhishTank: fornece informações relacionadas com tentativas de *phishing*;

- VXVault: fornece listas de endereços IP e de FQDNs que estão envolvidos em atividades maliciosas.

As fontes de informação irão sustentar o IntelMQ e este, por sua vez, fará a remoção de eventos duplicados (*deduplication*), classificação, etiquetagem e filtragem de todos os dados recebidos.

Para a recolha de informação direcionada às entidades-alvo foi necessário, primeiramente, observar e mapear (*scouting / mapping*) estas entidades, de modo a obter os atributos / campos mais relevantes de cada entidade, tal como, endereços IP públicos (IPv4 e IPv6), FQDNs, entre outros. Após a identificação destes campos, foi possível direcionar a filtragem de informação utilizando o IntelMQ.

Para o *scouting* e *mapping* de cada entidade-alvo será utilizado o Maltego. Este trata-se de um sistema interativo de *data mining*, que constrói gráficos direcionados para a análise de correlação de dados (*link analysis*). O Maltego é utilizado para investigações online para encontrar relações entre diferentes pedaços de informação de diversas fontes localizadas na Internet.

Com o Maltego é possível criar *"case-files"* através de uma representação gráfica de cada entidade-alvo com os atributos mais relevantes de cada uma. Estas representações gráficas contêm agregações e relações de informação relativas a eventos direcionados à entidade-alvo.

Após o *scouting* e *mapping* e após a definição dos atributos mais relevantes de cada entidade-alvo, foi possível direcionar a recolha de *Cyber Intelligence Events* no IntelMQ. Este foi configurado através de um programa Ruby (desenvolvido ao longo deste projeto), denominado *"intelmq_configurations_generator.rb"*. O programa utiliza os metadados recolhidos durante a fase de *scouting*, de modo a, reescrever os ficheiros de configuração

do IntelMQ e gerando regras de filtragem, consoante a gama de endereços IP e / ou FQDNs do conjunto de entidades-alvo. Os eventos são filtrados de acordo com as características de cada entidade-alvo, e são enviados em tempo-real para ficheiros que representam o universo de eventos de cada entidade, assim como para a plataforma Hidra, que permite a análise forense dos eventos filtrados.

Palavras-chave: ciber segurança, ciber inteligência, eventos, incidentes, ciberataques, fontes de informação

Abstract

Over the years to the present day, the concern around Cyber Security in organisations has increased substantially, because, most of the organisations rely on digitized information and share large amounts of data across the globe, becoming easier targets for many forms of attack. A cyber attack can damage an organisation's name and reputation, and can also result in loss of competitive advantage, create legal / regulatory noncompliance and cause steep financial damage.

In order to avoid a possible attack in an organisation's infrastructure is necessary to develop a strategy for the collection, analysis and correlation of information. In this way, it will be possible to better understand the type of attacks that an organisation might be exposed to, and in the future predict these attacks.

The main goal of this project was the development of a scalable architecture to collect, deduplicate, classificate, tag, filter and analyse Cyber Intelligence Events and applying them into the context of organisations, business units, infrastructures, assets and actors.

It was necessary to collect and process security feeds, which are streams of information consisting of facts and evidences that a certain event occurred, such as IP addresses or domains that were involved in malicious activities - Cyber Intelligence Events.

The Cyber Intelligence Events were filtered, according to a group of attributes composed by IP addresses, URLs and FQDNs of seven target-entities. The events were sent to an analysis platform, allowing to forensically analyse them, to better understand what, how and who performed the attack.

Keywords: cyber security, cyber intelligence, events, incidents, cyberattacks, intelligence feeds

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Over the years, the concern with Cyber Security (the protection of systems, networks and information in a cyberspace) in large companies has increased. Nowadays, most of organisations rely on digitized information and share large amounts of data accross the globe, becoming easier targets for many forms of attack. Consequently, a cyber attack may harm a company's name and reputation, resulting in loss of competitive advantage, creating a legal / regulatory breach and causing financial damage.

In order to avoid a possible attack in the organisation's infrastructures, it is necessary to take precautionary measures, in other words, to carry out an analysis and risk management, regarding the threats that a company is exposed to and thus outline a strategy in order to minimise, mitigate and / or anticipate attacks.

Portugal Telecom, also known as PT Portugal or PT Group, being the largest telecommunications service provider in Portugal, is aware of the concern with Cyber Security. PT Portugal has a department dedicated to security and privacy of information, the Direção de Cyber Security and Privacy (DCY).

For the purpose of having a resilient cyber protection in PT's infrastructures / assets, the DCY is divided into three different operations, as seen in figure 1.1:

- *Cyber Security Governance*: responsible for the management of DCY's Cyber Security operations / programs, incorporating goals for each operation;
- *Cyber Security Operations*: responsible for incident response;
- *Cyber Watch*: responsible for proactive risk analysis, in other words, identification of assets that may be affected by a cyberattack and continuously monitor the risk environment.

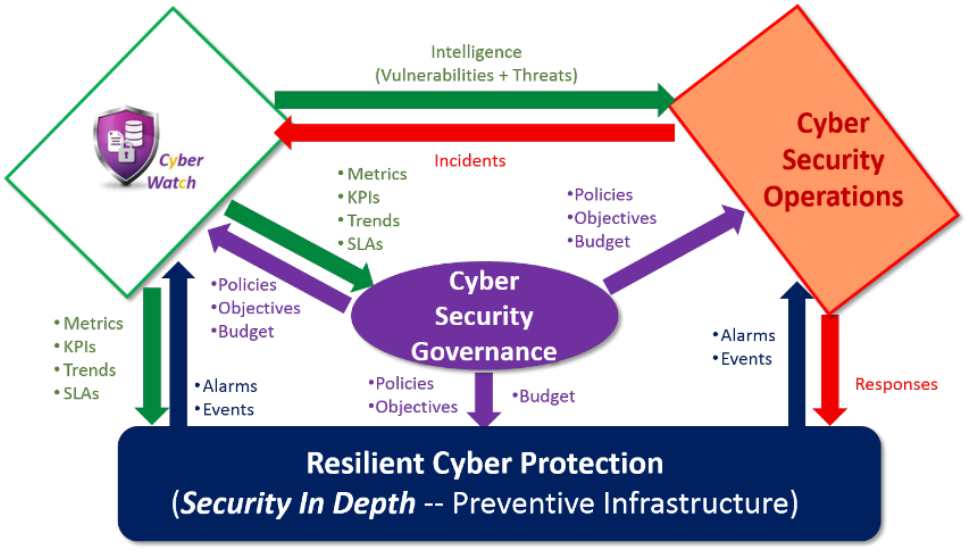

Figure 1.1: DCY's operations

The project "**Unified Cyber Threat Intelligence**" is part of the Cyber Watch operation of DCY and is being developed at Portugal Telecom, within the scope of PEI (Projeto em Engenharia Informática) subject of MI (Mestrado em Informática) of FCUL (Faculdade de Ciências da Universidade de Lisboa).

Cyber Watch is composed by different areas as seen on figure 1.2, in this way, the proactive risk analysis will be as effective as possible. The focus of this project inside Cyber Watch is within Cyber Intelligence.

Figure 1.2: The Different Areas of Cyber Watch

Cyber Intelligence is a type of information which provides an organisation with decision support, leading to a strategic advantage by supplying users with constantly updated information on possible sources of attack.

In the context of Portugal Telecom, Cyber Intelligence breaks down into different branches. The most important ramifications for this project are **risky sources**, which refers to the ”Universe of Events” consisting in all the events that happened on Internet in general, and **compromised targets**, which refers to events that happened within an organisation.

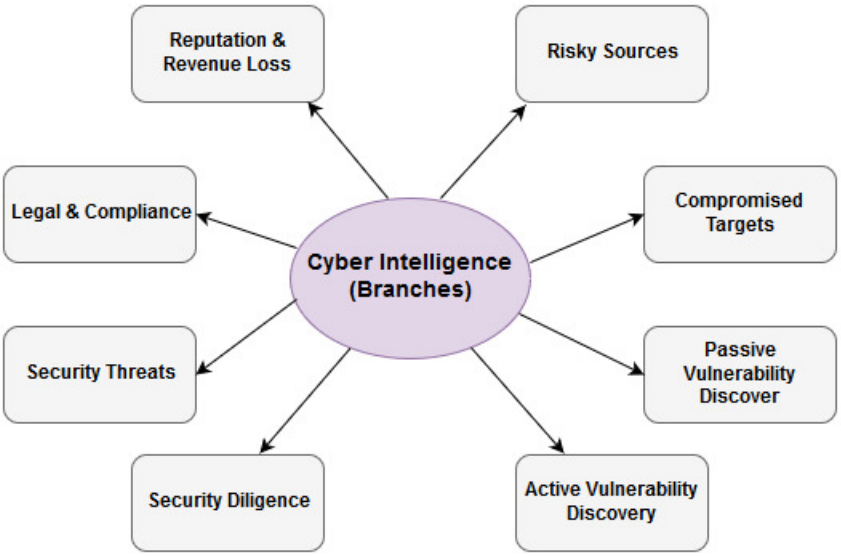

Figure 1.3: Cyber Intelligence Branches

The main goal of this project is the implementation of a scalable architecture for the collection, analysis, deduplication, classification, tagging and filtering of Cyber Intelligence events, being applied in the context of organisations, business units, infrastructures, assets or actors.

1.1 Motivation

Nowadays, the concern with Cyber Security in large companies is a reality, not only because there are legal obligations, required controls, criminal responsibilisation and in some cases fine payments, but also for the impact that an attack, intrusion and ilicit access with success, can bring upon the image and business of a company.

In order to mitigate these problems, organisations need mechanisms that collect and correlate information, regarding a possible attack of the infrastructures, credentials leak, vulnerable systems, sensitive documentation and others, thereby allowing a reduction of the time detection, investigation and mitigation of the incidents assigned by the security analysts.

At Portugal Telecom there is a great concern with Cyber Security, and regarding the problems mentioned previously, PT has a department dedicated to security and privacy of information, the Direção de Cyber Security and Privacy (DCY).

This department is divided in different sections and operations providing to Portugal Telecom a resilient cyber defense. However, to better mitigate any compromise in the infrastructures, it is fundamental to understand the attacks which an organisation is exposed to, which means analyse past events that occurred whithin and outside the organisation.

Therefore, the main motivation for this project was the availability of a system solution to facilitate the realisation of the analysis of past events, allowing any organisation to build better defending mechanisms to avoid and / or mitigate possible attacks.

1.2 Goals

The main goal of this project is the implementation of a scalable architecture for the collection, analysis, deduplication, classification, tagging and filtering of Cyber Intelligence Events, being applied in the context of organisations, business units, infrastructures, assets or actors, within the scope of Portugal Telecom. Thereafter, it will be made an analysis of the filtered Cyber Intelligence Events, in order to better understand the type of attacks that the organisation is exposed to, and to verify if the developed solution can filter the events correctly, according to a set of characteristics of each organisation.

To achieve the goal described in the previous paragraph, firstly it will be necessary to ”map” and ”scout” a group of target-entities, in order to have their most relevant attributes, such as IP addresses, URLs and FQDNs. The second part of this project will be focused on the configuration of a Cyber Intelligence Tool for the collection, processing and filtering of security feeds, which are streams of information consisting of facts and evidences that a certain event occurred. Finally, the filtered data will be analysed, to better understand the type of events that occurred regarding each target-entity.

1.3 Contributions

This section describes all the steps taken to achieve the main goal of this project.

As mentioned previously, the main goal of this project is to collect, filter, analyse and treat Cyber Intelligence events regarding certain target-entities.

The first step to achieve the main goal was to define which entities would be used for this project. These entities are companies that belong to Altice Group (Portugal Telecom owner) or are PT’s clients. It was chosen four entities: Entity-A, Entity-B, Entity-C and Entity-D.

The second step was to ”map” and ”scout” each target-entity, in order to discover all their public information on the Internet, such as, IP addresses, Fully Qualified Do-

main Names, Domains, URLs and others. With this step was possible to obtain metadata representing uniquely each entity. Then, the collected metadata served as input for the IntelMQ Configuration Manager (ICM), which is the program responsible for the automatic configuration of the Cyber Intelligence Tool IntelMQ, this tool collects and filters Cyber Intelligence events.

The IntelMQ Configuration Manager generates all the necessary files to configure IntelMQ, to filter the events that belong to the target-entities. All the configuration files generated by the program were made taking into account the metadata that compose each target-entity.

After the generation of the configuration files, IntelMQ is set to filter all the Cyber Intelligence events that are related to the target-entities. The filtered events are then sent to an analysis platform, where they are stored. The events are also saved into separate files, one per target-entity. In the analysis platform, it is possible to forensically examine the filtered events and correlate them.

1.4 Structure of the Document

This document is organised in the following way:

- Chapter 2 – Context and Related Work. The purpose of this chapter is to give a context to this project, by introducing key terms, methods of collection and analysis of Cyber Intelligence and its context inside Portugal Telecom.
- Chapter 3 – Cyber Intelligence Tools Analysis. This chapter describes and analyses the tools that were used for this project.
- Chapter 4 – IntelMQ Configuration Manager Solution Design. This fourth chapter presents the solution for the filtering of Cyber Intelligence events that are related to the target-entities. It is explained the requirements for the IntelMQ Configuration Manager and how the configurations for IntelMQ are produced.
- Chapter 5 – Implementation. In this chapter it is described in detail the development of the IntelMQ Configuration Manager, which is a Ruby Script composed of several methods, where each of them is responsible of the generation of configuration sets for IntelMQ.
- Chapter 6 – Results and Evalution. The sixth chapter presents the results obtained by the filtering of the Cyber Intelligence events, it is explained how the filtering was done (conditions and machine specifications) and the final result of the filtering for each target-entity.
- Chapter 7 – Conclusion and Future. Finally, this chapter describes all the conclusions, as well as future work to improve the current project.

Chapter 2

Context and Related Work

This chapter explains the context and related work of this project. It also introduces key terms and aspects for the project. It is described what is Cyber Intelligence, how it is collected and treated and what type of data it produces. It is also explained how Cyber Intelligence is applied in the context of Portugal Telecom and the different tools that exist to collect and treat Cyber Intelligence and from where this information is collected. Therefore, this chapter is divided in the following way.

Section 2.1. explains what is Cyber Intelligence and from where this term originated. Section 2.2. describes how Cyber Intelligence is collected and what types of output it generates. Section 2.3. explains what is Counter Intelligence. Section 2.4. describes what is Threat Intelligence. Section 2.5. explains how Cyber Intelligence is treated and processed and its lifecycle. Section 2.6. explains what are Cyber Intelligence Feeds, giving some examples. Section 2.7. describes some Cyber Intelligence Tools and explains their purpose. Section 2.8. explains the meaning of Cyber Intelligence at Portugal Telecom.

2.1 Cyber Intelligence

There is no definitive definition for Cyber Intelligence. However, there are concepts, theories and ideas for describing it. To understand Cyber Intelligence it is necessary to realise that intelligence tactics, techniques and procedures, as well as several types of operations existed long before cyber space was conceived [3].

In a military context commanders want to know the intent of the adversary, for two major reasons: (1) to make better strategic choices on the battlefield; (2) to prepare correctly for an attack. The definitions used by several government and military organisations may serve as the best foundation for understanding Cyber Intelligence [24].

According to the U.S. Department of Defense (DoD) the definition of intelligence is:

1. *The product resulting from the collection, processing, integration, evaluation, analysis, and interpretation of available information concerning foreign nations, hostile or potentially hostile forces or elements, or areas of actual or potential operations*

2. *The activities that result in the product*

3. *The organisations engaged in such activities* [5]

From the above definition it is possible to understand that DoD views intelligence as a product, the activities that lead to this product, and the organisations performing the activities. However, in general, most uses of intelligence will not include goals defined by foreign nations. The most simple definition can be presented as: *Intelligence is both a product and process from collecting, processing, analysing and using information to meet an identified goal* [24].

This definition can be applied to Cyber Intelligence, however, is very important to make sure that the data meets some goal or purpose [24].

There are several types / disciplines within Cyber Intelligence, these may vary according from where (source / origin) and how (method) the information was collected.

Cyber Intelligence can be collected from Open Source Intelligence (OSINT), Social Media Intelligence (SOCMINT), Human Intelligence (HUMINT) or intelligence from the deep and dark web. The collection method is an advanced process that enables the organisation to gather valuable insights, based on the analysis of contextual and situational risks and can be directioned to the organisations specific threat landscape, its industry and markets [24].

The following sections introduce and explain the different types of Cyber Intelligence, as well as the different methods of data collection.

2.2 Collection Operations

This section describes different types of collection operations of Cyber Intelligence, as well as, the types of data that are obtained.

There are many ways to collect data. According to the purpose of an organisation, the collection of Cyber Intelligence must be adapted to meet certain goals. However, the most important aspect of Cyber Intelligence Collection Operations is correctly identifying the right sources of data and making sure the data is valid.

2.2.1 Types of Collection

There are three main categories of collection operations:

- Passive - data collected on information systems or networks. This could be, for example, monitoring internal activities within an organisation, such as capturing internal network traffic or collecting system logs.

- Hybrid - data shared from other networks or information systems or collected from networks designed to attract adversaries. This could be, for example, two identical

entities sharing information between them about a potential attacker, helping each other to be prepared in case of an attack. Another example are honeypots produced to entice adversaries into interacting with them.

- Active - data obtained from external networks or information systems under the influence of an adversary. It must be taken into account, that sometimes networks or information systems under the influence of adversaries, might not actually be owned or controlled by them. For example, a Command and Control server being used to connect to malware. The server may belong to a victim while it is being used by the adversary [22].

2.2.2 Types of Data

After understanding the different types of data collection, it is very important to determine the type of data collected. There are three types of data:

- Raw Data - unvaluated data collected from a source that includes raw details such as IP addresses, network logs, or forum posts from a potential attacker.

- Exploited Data - data processed and analysed by another analyst which contains selected raw data. In other words, this type of data should include analysis on what the data means or indicates, for example, malware or computer campaign reports with technical information and analysis on a threat.

- Production Data - data finalized into a report for dissemination purposes that includes limited or no raw data. This type of data is intended for awareness of a reader / customer or for suggested actions. An example would be advices given to users or Intrusion Detection System (IDS) signatures ready-made for deployment [22].

2.3 Counter Intelligence

The main goal of Cyber Counter Intelligence (CCI) is to prevent, deter, defeat, or manipulate the adversary from conducting intelligence operations on a certain individual, or those who the individual protects, or the individual's organisation including its operations [21].

There are two types of CCI: **Defensive** and **Offensive**.

Defensive Cyber Counter Intelligence is seen as actions taken to identify and counter adversary intrusions before they occur and also as the efforts in identifying and minimising the threat scenery. In other words, strengthen defenses and prevent intrusion [21].

The focus of Defensive CCI is to understand the adversary and reduce the threat scenery to which they might exploit. This type of approach produces reports and analyses that gives to defenders a complement to their network and information security [21].

Offensive Cyber Counter Intelligence is seen as interactions with the adversary to directly collect information about their intelligence collection operations or to mislead them. For example, in Offensive CCI fake personas can be used on online forums, to gather information about adversary intelligence collection operations, or to flip the adversary operators into double agents to infiltrate the adversary's operation or publish false reports and information to mislead opponent intrusion attempts [21].

These actions can be performed inside and outside the organisation's networks. An example of an Offensive CCI operation would be identify or alienate adversaries already in the organisation's network, by creating a honeypot. This honeypot could have files that the adversary might be interested in, but which contain fake or incorrect data, allowing to identify malicious actors on the network, if they try to access the honeypot [21].

2.4 Threat Intelligence

The term threat is often misused, particularly when a threat to one organisation may not be a threat to another. This happens because most of the organisations fail to identify threats, spending time and resources on less important areas, such as risk and vulnerability analysis, instead of mitigating and fixing problems [23].

A threat is a combination of three factors:

- **Intent** is a malicious actor's desire to target an organisation.
- **Capability** is their means to do so.
- **Opportunity** is the vulnerability the actor needs for performing an attack.

This combination can be summed up as "if an actor has the *intent* and *capability* but the organisation is not vulnerable or there is no *opportunity* present, then the actor is not a threat" [23].

Hereupon, Threat Intelligence is defined as *"evidence-based knowledge, including context, mechanisms, indicators, implications and actionable advice, about an existing or emerging menace or hazard to assets that can be used to inform decisions regarding the subject's response to that menace or hazard"* [14].

The key in Threat Intelligence is to identify what really is a threat for an organisation. If the organisation that is receiving threat intelligence does not know how to identify what information is applicable to them, then the whole operation will be useless [23].

An easier way to narrow the identification process to what really matters for the organisation is dividing threat intelligence in three categories:

- Timely Intelligence - latest global insights on attacks, attackers and targets;
- Relative Intelligence - goals, motivations and methodologies of adversaries, what they are after, why they are attacking and how they infiltrated organisations;
- Actionable Intelligence - helps focusing on better prioritising alerts and incidents, that pose a higher risk for the organisation [12].

If these categories are taken into account while developing a strategic Cyber Threat Intelligence program, it will allow security teams to quickly assess risk, prioritise alerts and threats that matter the most, minimise exposure to attack, and ultimately save time and money by increasing the efficiency of the security operations [12].

2.5 Treatment and Processing

Intelligence has a lifecycle, this lifecycle is a process in Cyber Intelligence that is very important to convert data into intelligence useful for the organisations. First of all, to determine what the organisation's requirements are, it should be chosen a plan and a direction, meaning, they must have a defined goal and intentions. It can be something very simple, such as, wanting to know the command and control servers of a piece of malware, to block it in the network, and to learn the type of information systems that the organisation's target uses, making possible to infiltrate them.

The second step, is to define where and how to acquire the data and information to be processed. This can be, honeypots, dark and deep web, firewalls, or others. Then, it is necessary to process and convert the collected information, into something possible to use, such as, being able to access and parse through the data that was collected, or converting it, to human readable information.

Afterwards, the collected data should be transformed into an intelligence product. This is achieved through analysis and interpretation, and thus is heavily dependent on the analyst. The report results should meet a defined intelligence need or goal from the planning and direction phase.

And finally, the finished intelligence product should be supplied to whom it may concern, but if a user cannot access the product or cannot use it, it means that the final result did not meet the organisation's goal. By following these steps, it is possible to gather a great start into understading Cyber Intelligence and using it as an advantage in case of an attack [24].

2.6 Cyber Intelligence Feeds

The word *feed* just by itself, means to nourish something and this meaning can be applied on the World Wide Web. A web feed, also known as data feed or news feed is a data

format that provides to users constantly updated content. There are content distributors which make this information available for users to subscribe it [27].

In the Cyber Security World there are also information feeds, normally called Cyber Threat Intelligence Feeds. These feeds provide actionable information on adversaries, being possible to build a stronger Cyber Defense. With the evolution of the cyber space, it has been necessary to invest and create many sources of information on threat actors [44].

Nowadays, there are several sources of information in the Cyber Intelligence community. These sources rely on anti-malware, firewalls, and other "plug and play" platforms but they do not encompass the entirety of network security today. Quality threat intelligence feeds deliver the aggregation of multiple sources which only present a true portrayal of threats and vulnerabilities when examined all together [28].

Some examples of Cyber Threat Intelligence Feeds are the following:

- AlienVault: Provides multiple sources, including large honeynets that profile adversaries;
- BitSight Cyberfeed Stream: Provides information on compromised IP addresses, malware server names, destination IP addresses, ports, host names and others;
- MalwareDomains: Provides a list of malware domains;
- PhishTank: Provides phishing data;
- Shadowserver: Provides different types of data which is filtered appropriately for responsible areas.

The Cyber Threat Intelligence Feeds collect different types of data, which can be filtered according to different attributes, known as Indicators of Compromise (IOC). Indicators of Compromise are artifacts observed on a network or in an operating system that with high confidence indicate a computer intrusion.

There are Feeds that are specific for certain IOCs, for example, they only list IP addresses that were involved in malicious activities, or they can have information in many other IOCs, such as:

- FQDN (Fully Qualified Domain Names) which is the complete domain name for a specific computer, or host, on the Internet;
- Port, the port used on a malicious activity;
- URL (Uniform Resource Locator), which is the source or destination web address of a malicious activity;

- CIDR (Classless Inter-Domain Routing), which provides a network address that may be the source or destination of a malicious activity;
- Protocol application, which is the protocol used on a malicious activity;

  In addition to the IOCs, there are also security Feeds, which provide information on:

- Country, which is the Country name from where originated the malicious activity or the destination for the attack;
- City, which is the City name from where originated the malicious activity or the destination for the attack;
- ASN (Autonomous System Number), which is a unique number that is available to identify an autonomous system and which enables that system to exchange exterior routing information with other neighboring autonomous systems. This can be the source or destination ASN of the malicious activity;
- AS Name, the name associated with an Autonomous System, which can be the source or destination AS name of the malicious acitivity;
- Contact, which usually is the e-mail address of the attacker and / or the victim;
- Malware name, which is the name of the malware used on the malicious activity.

2.7 Cyber Intelligence Tools

There are available several tools (open-source and paid) to help security analysts to collect and treat Cyber Threat Intelligence. These tools support an analyst to search and understand different types of data. They allow any organisation to build resilient defense mechanisms and to be prepared for an attack even before the attack occurs. The best tactic for a resilient cyber defense is to combine different tools and collection mechanisms.

Some examples of Cyber Intelligence Tools are the following:

- Research tools: An example of a research tool is the website *virustotal.com*. Analysts turn to this type of resources for quick reference regarding many types of Indicators of Compromise. Virustotal is also a community where analysts report suspicious / malicious indicators, isuch that this service has historical data and current data, as well [4].
- SIEM (Security Information and Event Management): A SIEM is a powerful tool which allows analysts to monitor their organisation's network traffic in real time, allowing Incident Response teams to react to incoming threats [4].

- Threat Intelligence Provider: Organisations sometimes hire outside services / companies for their Threat Intelligence collection requirements, monitoring for ongoing and developing threats that matter to them. These include social media monitoring, hacktivism campaigns and CVE (Common Vulnerabilities and Exposures) awareness, regarding the concerns of an organisation, their business and brand. However, there are some organisations which prefer to have internal teams focus on this task. An example of a tool for this purpose is *Recorded Future*. This tool allows analysts to access the collected information, while also offering a suite of powerful search functionalities that provide an internal Intel team the ability to customize and automate searches[4].

- Network Traffic Analysis Framework: An example of a network traffic analysis framework is *Bro*, which is used by a wide variety of security professionals. *Bro* is a network monitoring framework that can be used for detecting network intrusion, collecting network measurements, and generating an extensive set of log files that records a network's activity in high level terms. These logs include not only a comprehensive record of every connection seen on the network, but also application layer transcripts such as all HTTP sessions and their requested URIs (Uniform Resource Identifier), key headers, MIME (Multipurpose Internet Mail Extensions) types, and server responses [4].

- Disassembler: Malware is an increasing concern among organisations and it is constantly being refined and improved. Reverse engineering malware is a process that Incident Response teams can use to identify how malicious a threat is, as well as give the team insight into how to defend against similar attacks in the future. *IDA Pro* is a great disassembler that explores binary programs and creates maps of a malicious file's execution [4].

- Web Proxy: There are tools that analyse inbound traffic in a safe environment, in order to prevent infection when a user accidentally or inadvertently visits a website that may be hosting malicious content. An example of tool for this purpose is *Burp*. It provides to an analyst a safe line of control for the inspection of traffic interacting with a network, making it tougher for threats to make their way in [4].

- Cybersecurity Platform: A security platform helps organisations to identify, manage, and block threats faster. It gives the possibility for a Response Team to integrate multiple tools in just one platform. An example of a Cybersecurity Platform is *ThreatConnect*. It allows users to customise and import threat data feeds to their instance, as well as the ability to join other like organisations in Communities that share similar threat data [4].

2.8 Cyber Intelligence at Portugal Telecom

As mentioned previously in this chapter, there is no definitive definition for Cyber Intelligence and each company has their own concept of Cyber Intelligence according to the company's goals and intervation areas. Cyber Intelligence must be used wisely, in order to prevent / mitigate any attacks which an organisation might be exposed to. At Portugal Telecom Cyber Intelligence is a combination of different areas / subjects, as shown in figure 2.1.

A detailed explanation of the different areas of Cyber Intelligence at Portugal Telecom is given, as follows:

- Reputation and Revenue Loss: every company has a brand to defend and to maintain, in order to provide a reliable service and confidence to its costumers. However, an organisation might be a victim of *brandjacking*, which is someone falsely implying some kind of association with the brand. An example of *brandjacking* are *phishing* activities, where the perpetrator actively promotes themselves as a legitimate brand representative. This type of attack has two main negative effects, firstly, if the brand abuse was an attempt to capture sales for a competing product or service, it will lead to revenue loss. Secondly, if the brand abuse was carried out with the goal of damaging the brand's image, then this results as reputation loss [40].

- Legal and Compliance: an organisation must be aware and understand its legal obligations, regarding cyber security. It is very important to protect personal data from unauthorised access, damage, loss or disclosure. The company must ensure a level of protection that is appropriate, taking into account the harm that may be caused to individuals in the event of a data security breach and the nature of the data. According to the Act on the Protection of Personal Data [31], it is not required that organisations prevent cyber security breaches from occurring, but to take all the appropriate measures to protect the data. Otherwise, there will be a legal breach, if it is proven that the organisation did not take the appropriate measures to secure the data [6].

- Security Threats: *hackers* (someone who, with their technical knowledge, uses bugs or exploits to break into computer systems) are relentlessly inventive and are constantly evolving, in order to find new ways to steal and harm. These might include *hacktivism*, which is the insurgent use of computers and computer networks to promote a political agenda or a social change. In other words, it is the act of hacking, or breaking into a computer system, for a politically or socially motivated purpose. These type of threat can, as mentioned previously, damage the brand of an organisation. For example, an individual might leave a highly visible message on the home

page of a company that gets a lot of traffic or which incorporates a point-of-view that is being opposed [48] [38].

- Security Diligence: for every company, cyber security weaknesses represent a high risk for their operations, reputation and business. Security diligence allows to verify the overall health and hygiene of the organisation. There should be a team responsible for finding problems in the security of the company, making sure that there are not any information leakage, unmanaged file sharing, or exposed credentitials. If it is given the correct diligence to the network of the organisation, the data and the network itself it will be more secure [42] [20].

- Active Vulnerability Discovery: in general, the active vulnerability discovery allows to identify weaknesses throughout the network of an organisation, such as ports that could be accessed by unauthorised users, or software missing the latest security patches, helping to ensure network compliance with the company's security policy [30].

  - Manual Penetration Testing: this type of test is done by an expert engineer, and its focused on finding vulnerabilities or any type of risk in a machine / network [45].
  - Automatic: this test is much faster, efficient, easy, and reliable. It also tests the vulnerability and risk of a machine / network, but its automatic, meaning that this technology does not require any expert engineer. There are many tools that can be used for this purpose, such as Nessus or OpenVAs [45].

- Passive Vulnerability Discovery: this type of vulnerability discovery identifies the active operating systems, applications and ports throughout a network, monitoring activity to determine the network's vulnerabilities. However, while passive vulnerability discovery provides information about weaknesses, it can not take action to resolve security problems. It is checked the current software and patch versions on networked devices, indicating which device is using software that presents a potential entrance for attacks, and reference this information against public databases containing lists of current patches [30].

- Compromised Targets: every organisation should have mechanisms to identify attacks, to understand if their network is infected, or if there are any machines making unsolicited communications. For example, there is a Cyber Intelligence Feed, named ShadowServer which provides reports of malicious activities on the network of an organisation. With these reports is possible to analyse the type of attacks that happened within the organisation, and identify which machines / devices that were infected [39].

- Risky Sources: as mentioned in section 2.6, there are several Cyber Intelligence Feeds, which provide actionable information on adversaries. Mainly, they provide lists of Cyber Intelligence events, which are composed by IP addresses, or domain names used as command and control communication channels. These events occurred on the Internet and the Cyber Intelligence Feeds share them, this way many organisations are informed of what has happened outside of the company and can be prepared in case of an attack [17].

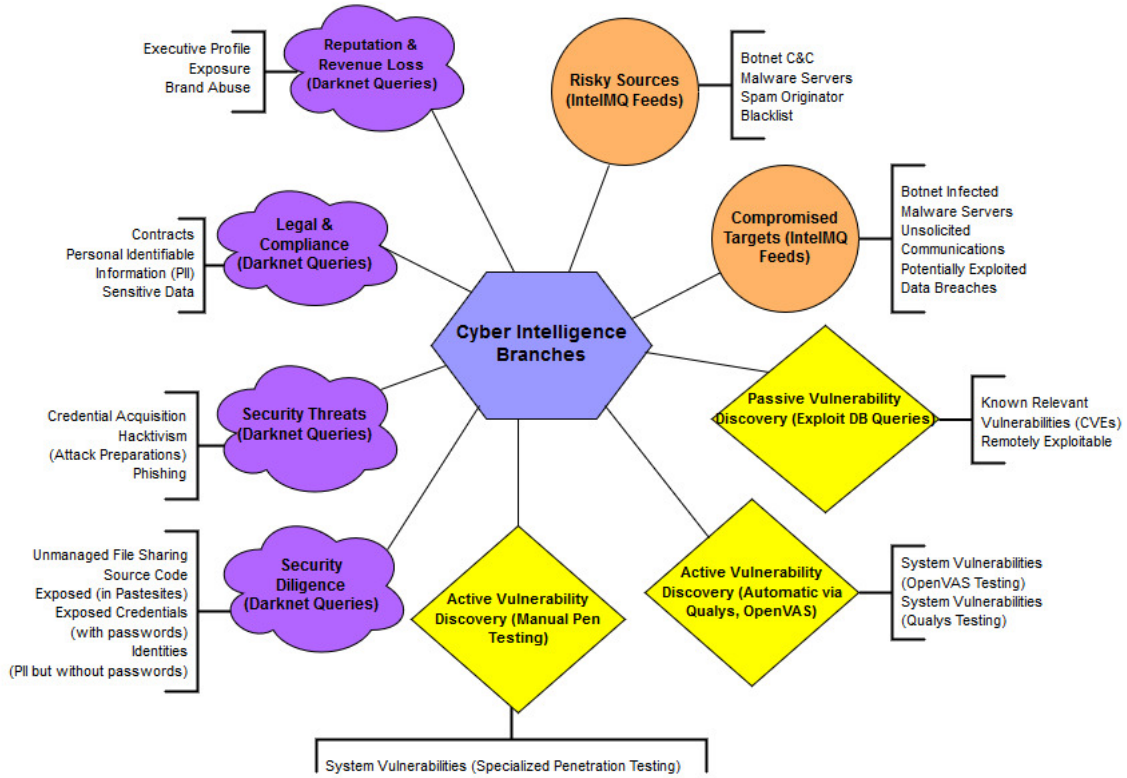

Figure 2.1: Cyber Intelligence at Portugal Telecom

Summarising, Portugal Telecom divides Cyber Intelligence into different sections, where they complement each other. This way is possible to understand what is happening within and outside of the organisation, being possible to build a better protection in the long run and avoid / prevent any form of attack to the company's infrastructures, data and assets.

2.9 Conclusion

This chapter introduced the concept of Cyber Intelligence, its origin, the different types that exist, how and from it is collected and how it is treated. Each type of Cyber Intelligence collection produces different data, and according to the type of data it should be treated in a specific way, in order to make the most of the collected information. Cyber

Intelligence can be collected from different sources on the Internet, and it is possible to filter information, according to what it is most relevant to the organisation. There are available different Cyber Intelligence Tools to collect data from the sources of information on the Internet, helping security analysts in the analysis and treatment of the obtained Intelligence. At Portugal Telecom, Cyber Intelligence is divided into different branches, and the combination of these different fields is what allows to the organision to prevent and / or mitigate any form of attack.

The following chapter introduces and analyses the Cyber Intelligence Tools that were used for this project.

Chapter 3

Cyber Intelligence Tools Analysis

As mentioned in the previous chapter, there are innumerable Cyber Intelligence tools which help security analysts to collect and treat Cyber Intelligence events. As the main goal of this project is the collection, analysis, deduplication, classification, tagging and filtering of Cyber Intelligence events, it was necessary to use Cyber Intelligence Tools to achieve the goal of the project.

This project will be based mainly in two distinct tools, one for data collection and analysis and another to filter Cyber Intelligence events. Therefore, the purpose of this chapter is to describe both tools, their purpose, how they function, and the type of outputs that they return.

The Cyber Intelligence events that were collected and analysed are events that already occurred, thus belonging to the past. It is very important to understand what happened in the past, in order to build a more resilient solution and avoid future attacks.

"To know your future you must know your past". [George Santayana]

3.1 Data Collection and Processing

To accomplish the goal of this project, it was necessary to collect and filter Cyber Intelligence events. An event, in the context of PT Cyber Watch, is an observable fact or occurrence on the public Internet (involving IP addresses, FQDNs and URLs), where it occurred within a certain period of time.

The first step to reach the main goal was to collect public information from the Internet about the target-entities. To gather the information, the Maltego tool was used. This tool will be described on section 3.2.

This tool has the capability to query several sources on the Internet and find links and relationships between several pieces of information. With Maltego it was possible to find the most relevant attributes of each target-entity, such as public IP addresses, DNSs,

FQDNs and URLs, being possible to characterise each entity uniquely.

The collected metadata about each target-entity was then used to configure the tool IntelMQ responsible for the collection, deduplication, classification, tagging and filtering of Cyber Intelligence events. A detailed analysis of this tool will be provided in section 3.3.

IntelMQ collects Cyber Intelligence events from different feeds, open-source and paid, external and internal. The purpose of this tool, in the context of this project, is to filter events according to a set of characteristics of the target-entities. For example, if an event contains an IP address that belongs to one of the target-entities, then that event is going to be filtered and directed to the corresponding entity.

Afterwards, all the filtered events will be sent to a platform named HIDRA (High performance Infrastructure for Data Research and Analysis) which is an agile platform for long-term event processing and traffic pattern analysis. This platform was developed at Portugal Telecom by DCY / CSD (CyberSecurity Development and Integration) and is based on Elasticsearch (responsible for storing all the logs of a system), RabbitMQ (message broker), Kibana (a web interface that can be used for searching and viewing logs) and Ruby programs. The main goal of this platform is to enable the rapid and agile development and implementation of complex event processing systems. This platform is going to store the filtered events and will allow to perform a forensic analysis to them [47].

3.2 Maltego

Maltego is an interactive data mining tool that renders directed graphs for link analysis. The tool is used in online investigations for finding relationships between pieces of information from various sources located on the Internet. Its main focus is on providing a library of transforms (scripts which search for specific information on several sources on the Internet) for discovery of data from open sources, and visualising information in a graph format, suitable for link analysis and data mining [33].

The focus of Maltego is analysing real-world relationships between information that is publically accessible on the Internet. This may include footprinting the Internet infrastructure, as well as finding information about the people and organisation who own it [33].

Maltego can be used to determine the relationships between several entities, such as:

- People (Names, E-mail Addresses, Aliases);
- Groups of People (Social Networks);
- Companies;

- Organisations;
- Websites;
- Internet Infrastructures (Domains, DNS Names, Netblocks, IP Addresses);
- Affiliations;
- Documents and Files.

Connections between these pieces of information are found using open-source intelligence techniques by querying sources such as DNS records, whois records, search engines, social networks, various online APIs and extracting meta data.

Maltego provides results in a wide range of graphical layouts that allow for clustering of information, allowing relationships to be quickly and accurately, making it possible to see hidden connections, and even allowing to know if they are three or four degrees of separation apart [33].

Maltego provides to its users two types of products: Maltego Clients and Maltego Servers.

There are four types of Maltego Clients, each tailored for different purposes.

- Maltego XL: allows to visualise large data sets and is suited for people who need to show relationships between up to 1 million pieces of information.
- Maltego Classic: allows users to visualise up to 10 000 pieces of information and the relationships between them.
- Maltego CE (Community Edition): this client is the community edition of Maltego, offering the same functionality as Maltego Classic with some limitations.
- CaseFile: solves the offline intelligence problem. This client combines Maltego's graph and link analysis functionality. This tool allows analysts to examine links between offline data.

The main difference between Maltego Classic, Maltego XL and Maltego CE are the number of entities that can be returned from a single transform and the maximum number of entities that can be on a single graph.

CaseFile on the other hand is mostly used by analysts using offline data who do not need access to the standard transforms within Maltego [33].

Regarding the Maltego's Servers, there are three types:

- CTAS (Commercial Transform Application Server): this server includes all the transforms found on Paterva's (Maltego's Company owner) public server.

- iTDS (internal Transform Distribution Server): this server has a web-based front-end that makes it easy to manage, share and distribute custom-built transforms from a common point.

- Comms Server (Communication Server): A comms server gives the user the ability to share graphs and have multiple people work on a single graph at the same time [33].

Maltego servers can be deployed within an organisation, which means that instead of having the organisation's transforms running over Paterva's infrastructure, the organisation can host their transform servers on an infrastructure that they control. An internal server gives to an organisation the ability to integrate with their structured internal data and leverage internal processes, as well as the ability to distribute these transforms across their enterprise [33].

Figures 3.1 and 3.2 show examples of Maltego Graphs:

Figure 3.1: Maltego Graph Example 1 [13]

Figure 3.1 shows the relationship between two people and their social networks.

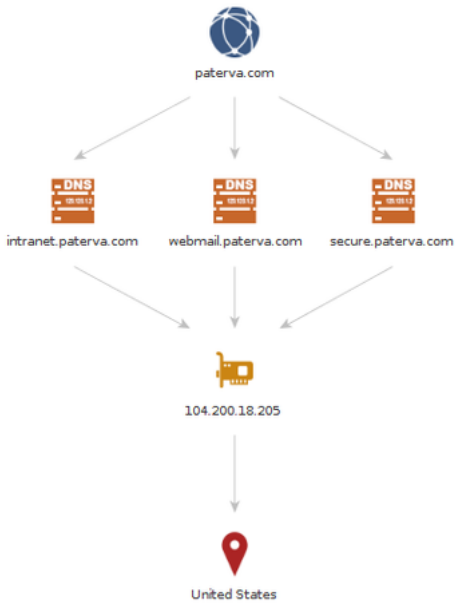

Figure 3.2: Maltego Graph Example 2

Figure 3.2 shows the relationship between a domain, DNS names, an IP address and a location.

3.3 IntelMQ

IntelMQ is a solution for IT (Information Technology) security teams, such as CERT (Computer Emergency Response Team) or CSIRT (Computer Security Incident Response Team), for collecting and processing security feeds using a message queuing protocol. These feeds are streams of information consisting of facts and evidences that a certain event has occurred. Its main goal is to give to incident responders an easy way to collect and process Cyber Intelligence events. IntelMQ can be integrated with other existing tools[9].

This tool has a modular structure based on bots (software applications that perform automated and repetitive tasks). There are four types of bots:

- Collector-Bots: these bots are responsible for the data collection from internal and external sources, the output that these bots return are reports consisting of many individual data sets;
- Parser-Bots: these bots parse the collected data by splitting it into individual events and giving them a defined structure;
- Expert-Bots: these bots enrich the existing events by adding extra fields of information, such as reverse records, geographic location information or abuse contacts;
- Output-Bots: these bots write events to files, databases, among others.

Each bot has one source-queue (apart from the *collector-bots*) and can have multiple destination queues (except for the *output-bots*). However, multiple bots can write to the same pipeline, resulting in multiple inputs for the next bot. Every bot runs in a separate process and they are uniquely identifiable by a *bot-id*. [9]

IntelMQ uses a concept named *botnet*. The *botnet* represents all currently configured bots, which are explicitly enabled. To add bots to the botnet or to configure them, IntelMQ uses four configuration files: *runtime.conf*, which contains all the bots configurations, *pipeline.conf*, which contains all the source and destination queues of every bot of the botnet, *harmonization.conf*, which specifies the fields for all message types, ensuring that the values are compliant with the ”harmonization” format, and *defaults.conf* where is specified the default values for all bots and their behaviour. The bots communicate between them, through the messaging queues specified in the file *pipeline.conf*.

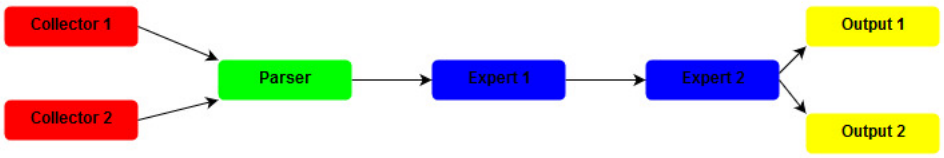

Figure 3.3: IntelMQ Botnet Example

Figure 3.3 represents an IntelMQ botnet. The red bots are the *collectors*, the green bot represents the parser-bot, the blue bots are the *expert-bots* and the yellow bots are the *output-bots*.

Figure 3.4 shows a template of the *runtime.conf* file and an example of a bot configuration in the same file. The first line of the configuration is always the bot name, which must be unique, the second line is where it is defined which group the bot belongs, it might be a *collector*, a *parser*, an *expert* or an *output*. The next line is the Python module to execute the bot, this field is followed by the description of the bot to specify its function in the botnet. Finally, it is defined the bot's parameters, these may vary according to the bot type. The parameters might be the feed URL from where the bot is collecting information, the feed name, the time interval that the bot must collect the information and many others.

Template:

```
{
    "<bot ID>": {
        "group": "<bot type (Collector, Parser, Expert, Output)>",
        "name": "<human-readable bot name>",
        "module": "<bot code (python module)>",
        "description": "<generic description of the bot>",
        "parameters": {
            "<parameter 1>": "<value 1>",
            "<parameter 2>": "<value 2>",
            "<parameter 3>": "<value 3>"
        }
    }
}
```

Example:

```
{
    "malware-domain-list-collector": {
        "group": "Collector",
        "name": "Malware Domain List",
        "module": "intelmq.bots.collectors.http.collector_http",
        "description": "Malware Domain List Collector is the bot responsible to get the report from source of
        "parameters": {
            "http_url": "http://www.malwaredomainlist.com/updatescsv.php",
            "feed": "Malware Domain List",
            "rate_limit": 3600
        }
    }
}
```

Figure 3.4: Runtime Configuration File Example

Figure 3.5 presents a template and a an example of a configuration of the file *pipeline.conf*. The purpose of this file is to configure the source and destinations queues of each bot, in other words, it configures the ”path” from where the messages (events) must pass in the pipeline until they reach the output of the botnet. Similar to the file *runtime.conf*, the first line of the configurations is always the bot-id, which is the unique identifier of a bot. The second line of the configuration is the the source-queue, which is the queue of the bot itself, while the destination queues are the bots queues to where the events must be sent. There are some rules related to the queues, according to the bot type. For example, the *collector-bots* can have several destination-queues, while *output-bots* can not have any, because they are the end of the pipeline.

In the example of the figure 3.5 the bot-id is *malware-domain-list-parser*, the source queue is the queue of the bot itself and the destination-queue is the bot *file-output*. As these configurations refer to the bot queues, it is added the word *”queue”* to the bot-id.

Template:

```
{
        ...
    "<bot ID>": {
        "source-queue": "<source pipeline name>",
        "destination-queues": [
            "<first destination pipeline name>",
            "<second destination pipeline name>",
            ...
        ]
    },
        ...
}
```

Example:

```
{
        ...
    "malware-domain-list-parser": {
        "source-queue": "malware-domain-list-parser-queue",
        "destination-queues": [
            "file-output-queue"
        ]
    },
        ...
}
```

Figure 3.5: Pipeline Configuration File Example

The files *runtime.conf* and *pipelime.conf* are in JSON format (JavaScript Object Notation), which is lightweight data-interchange format. It is easy for humans to read and write and it is easy for machines to parse and generate [18].

On IntelMQ, all messages (reports and events) are Python / JSON dictionaries. The key names and according types are defined by IntelMQ Harmonization Rules. These rules are meant to keep simplicity and consistency between the data and the data fields that compose an IntelMQ event.

A field is a key=value pair and for a clear and unique definition of a field, the key (field-name) must be defined, as well as the possible values. A field belongs to an event, which is a structured log record in the form key=value, key=value, key=value. The fields definition is made on the *harmonization.conf* file. This file follows the same rules as the *runtime.conf* and *pipeline.conf* files. Figure 3.6 shows an example of the *harmonization.conf* file.

Template:

```
{
    "<message type>": {
        "<field 1>": {
            "description": "<field 1 description>",
            "type": "<field value type>"
        },
        "<field 2>": {
            "description": "<field 2 description>",
            "type": "<field value type>"
        }
    },
}
```

Example:

```
{
    "event": {
        "destination.asn": {
            "description": "The autonomous system number from which originated the connection.",
            "type": "Integer"
        },
        "destination.geolocation.cc": {
            "description": "Country-Code accoriding to ISO3166-1 alpha-2 for the destination IP.",
            "regex": "^[a-zA-Z0-9]{2}$",
            "type": "String"
        },
    },
}
```

Figure 3.6: Harmonization Configuration File Example

To validate IntelMQ events these must fulfill certain requirements. The minimum requirements for an IntelMQ event to make sense are [17]:

- Feed name, from where IntelMQ collected the event;
- Type of event that was found, for example, spam or malware;
- Taxonomy of the event, for example, Abusive Content (it can be associated to spam) or Malicious Code (it can be associated to malware);
- Time source, the time reported by a source (feed);
- Observation time, the time a source bot saw the event.

Additionally, an IntelMQ event must contain at least one of the following fields:

- Source IP, IP address that initiated the connection;
- Source FQDN, DNS name related to a host that originated the connection;
- Source URL, refers to a malicious resource whose interpretation is defined by the abuse type. A URL with phishing as an abuse type refers to a phishing resource;

- Source Account, name of an account or e-mail address related to the origin of an event.

There are several types of feeds where the *collector-bots* can retrieve information, these feeds can be open-source or paid, external and internal, and they contain several types of data, such as, IP addresses that were involved in malicious activities [9]. For each feed the *collector-bo*t must be configured accordingly and it must be used the *parser-bot* which corresponds to the input feed.

Table 3.1 shows all the feeds that are on IntelMQ.

Table 3.1: IntelMQ Feeds List

Feed Name	**Description**
Abuse.ch	It provides information on malicious domains, IP addresses, ransomware family and ZeuS trojan
Alien Vault	URL: It contains an IP reputation database that determines which public IP addresses are involved in malicious activity. OTX pulses: The OTX community reports on and receives threat data in the form of pulses. An OTX pulse consists of one or more indicators of compromise (IOCs) that constitute a threat, a campaign, or an infrastructure used by a malicious actor. IOCs act as vectors for active threats to networks and computers.
Autoshun	It contains a list of IP addresses related to organised crime or malicious activities.
Bambenek	It contains lists of IP addresses, domains and DGA (Domain Generated Algorithms) related to malicious activity of CnC servers
Bitcash	Blocklist provided by bitcash.cz of banned IPs for service abuse, this includes scanning, sniffing, harvesting, and DoS attacks.
BitSight Ciberfeed Stream	It provides information on compromised IP addresses, malware server names, destination IP addresses, ports, host names and others.
Blocklist.de	It provides information on malicious IP addresses, Bots, Brute-Force Logins, FTP, IMAP, IRC Bot, Mail, SIP, SSH and Strong IPs
Blueliv CrimeServer	It is a platform that provides targeted cyber threat information and analysis intelligence for large enterprises, service providers, and security vendors. Crime Servers Data related to a server that has been used to perform some kind of malicious activity.

Continued on next page

Table 3.1 – *Continued from previous page*

Feed Name	Description
CI Army	It provides threat intelligence harvested from the CINS (Cargo Incident Notification System) consisting of IP addresses related to malicious activities
CleanMX	It contains information about Phishing websites and malwares.
Cymru	It returns a list of fullbogons (private and reserved addresses and netblocks that have not been allocated to a regional internet registry). They are commonly found as the source addresses of DdoS attacks.
DShield	It provides information on AS numbers, subnets and malicious URLs
Danger Rulez	It provides a list of IP addresses that tried to perform SSH brute-force attacks
DynDNS	It provides a list of ponmocup malware redirection domains and infected web-servers
Fraunhofer DGA	It provides a list of domain names that were dynamically created by malware using Domain Generation Algorithms (DGAs)
HPHosts	It provides a list of malicous hosts
Malc0de	It provides a list of domains and IP addresses involved in malicious activities
Malware Domain List	It provides a list of malware domains
Malware Domains	It provides a list of malware domains
MalwarePatrol Dans Guardian	It provides information on malware & ransomware URLs, C&Cs, IP addresses, malware hashes and DGAs
Netlab 360	It provides information on DGA families and Magnitude Exploit Kit
Nothink	It provides information on attacks against DNS honeypots and IP addresses that tried to connect to honeypots via SNMP, SSH and Telnet
OpenBL	It provides a list of IP addresses that tried to perform various types of Internet abuse
OpenPhish	It uses proprietary Artificial Intelligence algorithms to automatically identify zero-day phishing sites and provide comprehensive, actionable, real-time threat intelligence
PhishTank	It provides phishing data

Continued on next page

Table 3.1 – *Continued from previous page*

Feed Name	Description
ShadowServer	Shadowserver collects different types of data which is filtered appropriately for responsible areas (ASN, CIDR, Country Code or TLD).
Spamhaus	It provides a list of netblocks that are "hijacked" used for dissemination of malware, trojan downloaders, botnet controllers
Taichung	It provides a list of IP addresses that were involved in malicious activities
Turris Greylist	It provides a list of IP addresses that have tried to obtain information about services on Turris routers
URLVir	It provides a list of malicious hosts and IP addresses hosting malware
VXVault	It provides a list of IP addresses and Hosts that are involved in malicious activities

3.3.1 Intelmqctl

Intelmqctl is the main tool to handle an IntelMQ installation and it also handles the bots themselves. It can be used as a command line tool, as a library and as a tool by other programs. If called directly, it will print all outputs to the console.

With Intelmqctl it is possible to start / stop the bots, verify their status, consult their log files, restart and reload the whole botnet. This is all done by commands through a terminal.

For example, to start a bot with the ID `file-output`, it will be created a file with its PID (Process Identifier) in the directory `/opt/intelmq/var/run/[bot-id].pid` and the output in the terminal for starting the bot is the following:

```
> intelmqctl start file-output
intelmqctl: Starting file-output...
intelmqctl: file-output is running.
```

Figure 3.7: Intelmqctl Start Command [17]

Figure 3.7 shows the intelmqctl command to start the bot `file-output` and then it prints to the terminal the status of the bot. If the bot was already running it will not be started again and its status will be printed to the terminal, as shown in figure 3.8:

```
> intelmqctl start file-output
intelmqctl: file-output is running.
```

Figure 3.8: Intelmqctl Starting a Bot That Was Already Running [17]

To stop a bot, if the PID does not exist a SIGINT will be sent to the process. A SIGINT is the interrupt signal. The terminal sends it to the foreground process when the *user* presses *ctrl-c* [29].

After 0.25 seconds (the minimum time to understand if a bot is running) is checked if the process it is running. If not, the PID file will be removed. Figure 3.9 shows the stop command.

```
> intelmqctl stop file-output
intelmqctl: Stopping file-output...
intelmqctl: file-output is stopped.
```

Figure 3.9: Intelmqctl Stop Command [17]

If the bot was not running, then there is nothing to execute, as shown in figure 3.10:

```
> intelmqctl stop file-output
intelmqctl: file-output was NOT RUNNING.
```

Figure 3.10: Intelmqctl Stopping a Bot that Was Not Running [17]

To restart the botnet or a single bot the commands stop and start are applied consecutively, as the following figure shows:

```
> intelmqctl restart file-output
intelmqctl: Stopping file-output...
intelmqctl: file-output is stopped.
intelmqctl: Starting file-output...
intelmqctl: file-output is running.
```

Figure 3.11: Intelmqctl Restart Bot [17]

For the reloading of a bot, Intelmqctl sends a SIGHUP to the bot, which will reload the configuration. SIGHUP (”signal hang up”) is a signal sent to a process when its controlling terminal is closed [29].

Figure 3.12 shows the reload of the bot file-output.

```
> intelmqctl reload file-output
intelmqctl: Reloading file-output ...
intelmqctl: file-output is running.
```

Figure 3.12: Intelmqctl Reload Bot [17]

If the bot is not running, then it can not be reloaded, as shown in the following figure:

```
> intelmqctl reload file-output
intelmqctl: file-output was NOT RUNNING.
```

Figure 3.13: Intelmqctl Reloading of a Stopped Bot [17]

3.3.2 IntelMQ Manager

To manage IntelMQ there is available a graphical tool for it, called IntelMQ Manager, as shown in figure 3.14. This is a graphical interface to manage configurations for the IntelMQ framework. An IntelMQ configuration is a set of configuration files, which are *runtime.conf*, *pipeline.conf*, *harmonization.conf* and *defaults.conf*, however, for this project only the files *runtime.conf* and *pipeline.conf* will be configured. This interface describes which bots and processing steps should be run and in which order. IntelMQ Manager is an intuitive tool that allows non-programmers to specify the data flow in IntelMQ.

The interface allows its users to visually configure the whole IntelMQ pipeline and the parameters of every single bot. Users will be able to see the pipeline in a graph-like visualisation similar to the following figure:

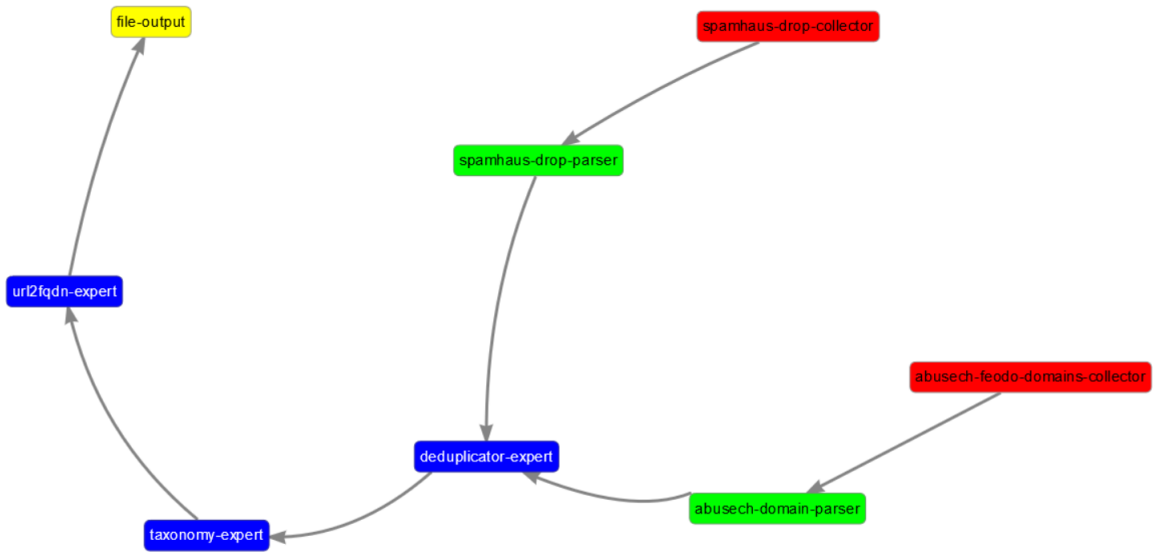

Figure 3.14: IntelMQ Manager Interface Pipeline View[17]

When a node (bot) is added or edited, the users are presented with a form with the available parameters for a bot. Then the parameters can be easily changed as shown in figure 3.15.

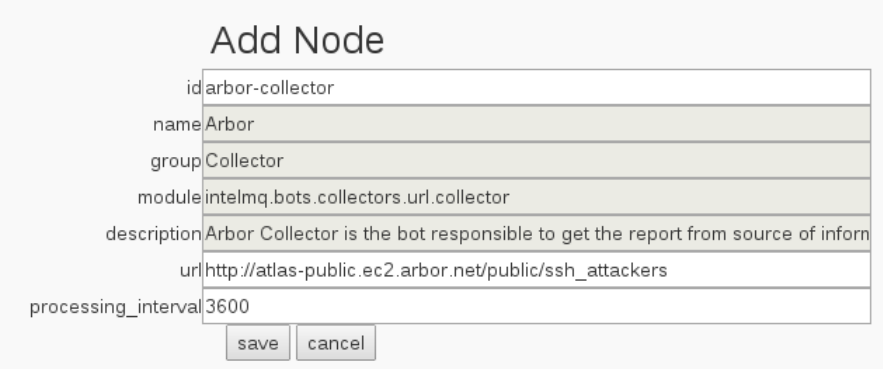

Figure 3.15: Form for Bots Configuration [17]

After editing the bots configurations and pipeline, it is possible to save them and the changes are automatically written to the correct files. In other words, the IntelMQ Framework originally comes with default configuration files, *runtime.conf* and *pipeline.conf*. These files already contain some bots configurations and it is possible to edit them and add more configurations ”by hand”, but with IntelMQ Manager there is no need to edit the files ”by hand”, because when a pipeline is created through IntelMQ Manager, this tool will edit these files and save the configurations automatically, being much more user-friendly.

The saved configurations are then ready to be deployed. The interface has a Management section where it is possible to observe the saved configurations. This section allows to start / stop the entire botnet or single bots and it also shows which of them are running / stopped.

The IntelMQ Manager Interface also allows to monitor the logs of individual bots or see the status of the queues for the entire system or for single bots.

3.4 Conclusion

This chapter described the main tools that were used for this project. It was also described how the data collection and processing between the two tools was made. Maltego was used to collect data on the target-entities. This tool used a set of transforms to find all the public information on the Internet about the entities, creating a graph with connections between the found data.

The metadata found by Maltego was used as input for IntelMQ, in order to filter the events that are related to the target-entities. IntelMQ uses several configuration files to form a botnet capable of collecting, tagging and filtering the events that are more relevant for this project. The events are collected from various Cyber Intelligence Feeds on the Internet and are filtered according to the information on the target-entities. IntelMQ can be managed through Intelmqctl, which uses the command line and is less user-friendly, but it also has a graphic interface, the IntelMQ Manager, which is more intuitive and more

user-friendly.

The following chapter will describe in detail how the solution for the filtering of the events was found.

Chapter 4

IntelMQ Configuration Manager Solution Design

This chapter explains the design of the solution that was found to filter all events, according to the range of IP addresses of the target-entities and their FQDNs and URLs, therefore this chapter is organised as follows.

The first section describes the requirements for the solution design, the second section describes the solution as a whole, while the following sections describe in detail all the steps to reach the goal of this project. The third section describes how the most relevant attributes of each target-entity were collect and organised. In other words, it explains how the IP addresses of the target-entities and their FQDNs and URLs were obtained. The fourth and fifth sections explain how IntelMQ filters the events according to the IP addresses, FQDNs and URLs of the target-entities.

4.1 Requirements

For this project it was developed a Ruby Script to configure IntelMQ automatically, the script configures IntelMQ to generate a specific botnet to filter Cyber Intelligence events, according to the attributes of the target-entities, which were obtained by Maltego.

The script must reach some requirements:

- It must be **dynamic**, it must not have static information, all the static information should be in configurations files;
- It must be **versatile**, if the botnet needs an update, for example, if we want to add a new bot to it, it should not to be necessary to make changes on the Ruby code;
- It must be **efficient**, it must not be time consuming nor consume computer resources;
- It must be **simple** and easy to understand, the script must be well documented, in order for others understand the code easily.

4.2 Solution Design

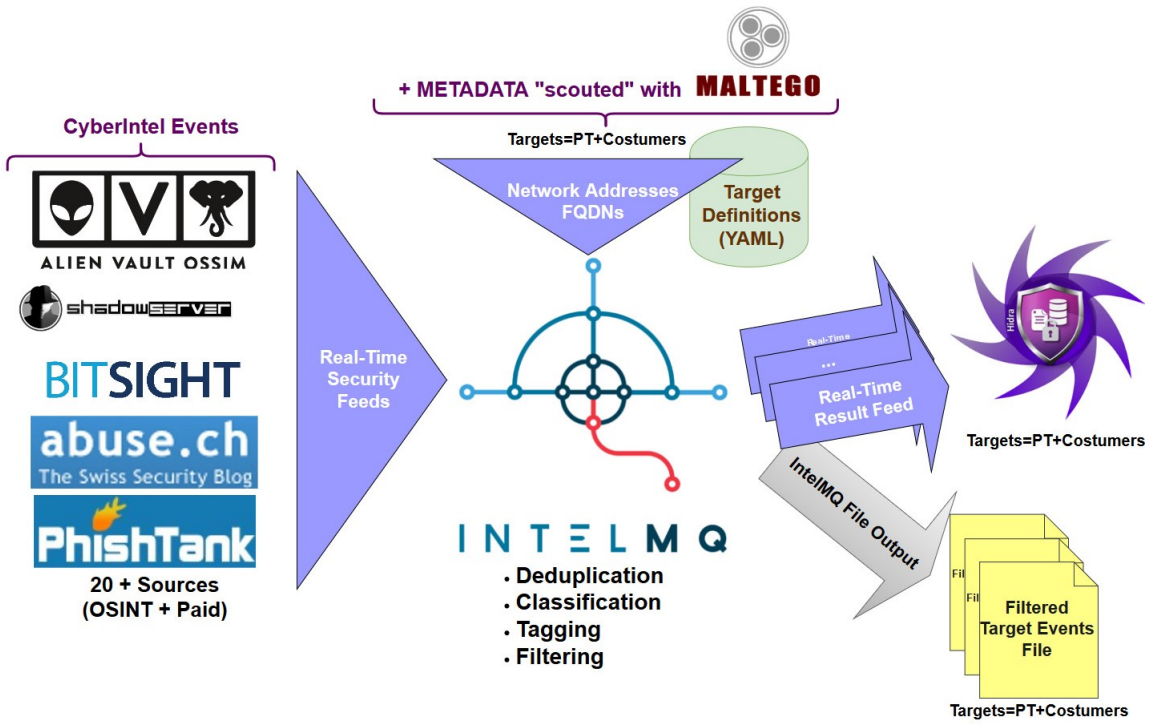

Figure 4.1: IntelMQ Configuration Manager Solution Design

Figure 4.1 gives an overview of the solution design for this project. At the top of the image there is the metadata obtained by Maltego on the target-entities, which is used to create YAML files. These files are then used as input to configure IntelMQ, which in turn, it collects information from different security feeds and performs the deduplication, classification, tagging and filtering of the Cyber Intelligence events. Lastly, the events are sent to HIDRA and to the corresponding file of each target-entity. This way the events are stored, thus allowing a forensic analysis (analysing and presenting facts about the events).

4.3 Maltego Metadata

To accomplish the goal of this project, firstly it was necessary to find the most relevant attributes of each target-entity, through scoutting and mapping the entities using Maltego.

As mentioned in the previous chapter, Maltego is used for online investigations for finding relationships between pieces of information from various sources, located on the Internet. In this particular case, Maltego was used to find public information about the target entities.

Each target entity has a public domain and it was from these domains that the online investigation started. In Maltego is possible to create different entities representing, in this case, the domains of each target entity and is possible to execute several transforms

on these domains. The transforms query several data sources, in order to obtain the information on the target entities. Each time a transform is executed, it is added to the graph representation, the corresponding information that the transform found, creating the links between the data.

The transforms executed on the target entities were: **DNS From Domain** and **Resolve To IP Address**. The DNS From Domain transform is a group of transforms, while the other two are single transforms.

The transforms that compose the **DNS From Domain** transform are:

- To DNS Name [Using DB]: this transform searches for the given Domain in the Robtex Database. Robtex contains billions of documents of Internet data, collected over more than a decade [32];
- To DNS Name - MX(mail server): this transform determines if an MX record exists for the given Domain. The MX record is the mail exchanger record and is returned as an MXrecord Entity. The IP address of this record gives a good indication of the network location of the target as most organisations keep their mail close to their network [32];
- To DNS Name - NS(name server): this transform determines if an NS record exists for the given Domain. The NS record is the name server record and is returned as an NSrecord Entity [32];
- To Domain Name - Zone Transfer: this transform attempts a zone transfer against all name server available for a domain [32];
- To DNS Name [attempt Zone Transfer]: this transform attempts a zone transfer on the Domain. If possible it extracts the Cnames and A records from the zone as DNS Name. If a zone transfer is possible then all the DNS names associated with the Domain are returned [32];
- To DNS Name [Find Common DNS Names]: this transform attempts to find DNS names for the specified Domain. This is done by testing a list of DNS Names and seeing if they exist, this list if configured on the transform [32];
- To DNS Name [Using Name Schema]: this transform will try several word lists, such as planet names, colours, TLDs and others, as DNS names. If it finds a match in a specific word list, it will try the entire word list. In this way it will try to determine the naming schema for the Domain [32];
- To Website [Quick lookup]: this transform will do a quick look up to verify if the DNS entry *www.domain* exists. This transform is useful when dealing with a large amount of domains and check quickly which of them have websites [32];

- To Website DNS [using Search Engine]: this transform will search for the domain name and then show the web sites where the domain name occurs [32].

Figure 4.2: Maltego Transform List: DNS From Domain

Figure 4.2 shows the list of transforms that compose the transform **DNS From Domain** on Maltego's application.

After executing the previous transforms, it was applied to the obtained result set the transform **Resolve to IP Address**. This transform resolves the DNS names into an IP address.

Table 4.1 shows the total of DNS names, Domains, IP addresses, MX records, NS records and Websites that Maltego found for each target-entity.

Table 4.1: Maltego's obtained data

Entities	DNS Names	Domains	IPv4 Addresses	MX Records	NS Records	Websites
A	80	5	88	1	5	30
B	25	3	18	1	2	8
C	1	1	9	4	4	2
D	257	20	78	1	4	72

After scouting and mapping the target-entities with Maltego and know their most relevant attributes, the metadata was used as input for a Ruby Program (which will be described on chapter 5). It was created YAML files with the metadata of each target-entity, in this case, it was created four YAML files, each one representing one target-entity. The format of the YAML files is shown on figure 4.3.

```yaml
entity_identification:

  name: entity-A

  classification_id: A

  output: #intelmq output bots
    A-file-output-1:
      name: File
      group: Output
      module: intelmq.bots.outputs.file.output
      description: File is the bot responsible to send events to a file
      parameters:
        file: /opt/intelmq/var/lib/bots/file-output/entity_A.txt
        hierarchical_output: false

  public_network_names:
    url:
      - http://www.example.com/
    fqdn:
      - entity-A.example.com
    domain:
      - entity-A.com

  public_network_addresses:
    ipv4:
      single_ip:
        - [redacted]
      range:
        - [redacted]
      subnet:
        - [redacted]
```

Figure 4.3: YAML File Structure Example

All the entity YAML files are composed by the following elements:

- name: which identifies the target-entity;
- classification id: the identifier used by IntelMQ to filter the events, according to the respective target-entity;
- output: the configuration of IntelMQ output-bot for the respective target-entity. For all the target entities it was used a *file-output-bot*, which means that the events related to the target-entities will go to a specific file that is specified in the bot configuration;
- public network names: this section is composed by lists of URLs, FQDNs and domains which were found by Maltego;
- public network addresses: this section is composed by lists of single IP addresses, ranges and subnets.

4.4 Case 1: IP Addresses First Pipeline

To fitler all the events according to the range of IP addresses of each target-entity, it was necessary to build a specific IntelMQ pipeline.

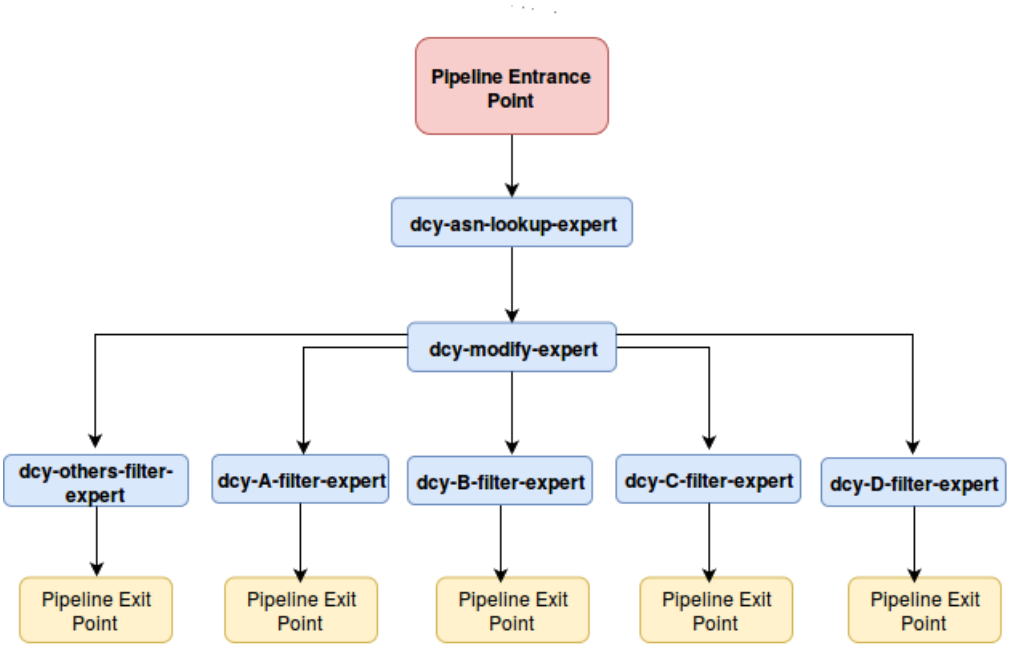

Figure 4.4: IntelMQ Pipeline to Filtrate IP Addresses

In figure 4.4 is possible to observe the first pipeline created to filter the events. The first part of the pipeline denominated by ”Pipeline Entrance Point” is composed by different collector bots and their corresponding parsers. As mentioned previously, the collector bots are responsible of the gathering of events in several Cyber Intelligence Feeds. Although, these feeds are the origin of events, the most important part of this project is how the events are filtered, according to the specifications of the target-entities.

The first expert bot used in this pipeline is the *asn-lookup-expert-bot*. This bot is responsible for adding AS numbers to the events . To make this bot work, it was necessary to build an ASN database. This database is composed by subnets that belong to the target-entities and for each target-entity it was assigned an unique AS Number, according to the Autonomous System (AS) Reservation for Private Use [16]. This means that there are 1023 different AS numbers, from the range 64512 to 65534 inclusive. If in the future there will be more than 1023 target-entities, it will be necessary to review the ASN algorithm. This bot will add to the events the fields *source.asn* and *destination.asn* if the IP addresses in the event are in the customised ASN database.

The ASN database is built with the single IP addresses, ranges and subnets that are in the YAML files of each target-entity, then the AS numbers are added to the database, according to the respective target-entity and their IP addresses. The AS numbers are attributed to each entity by the Ruby program, which will be explained in more detail in chapter 5. This database is composed by two columns, the first column contains all the IP addresses in the CIDR format and the second column contains the correspoding private AS number. Figure 4.5 shows an example of the ASN database.

```
; IP-ASN32-DAT-file   :
; Original source     : rib.20170125.1000.bz2
; Converted on        : Wed Jan 25 10:11:38 2017
; Prefixes-v4   : 681163
; Prefixes-v6   : 0
;
[redacted]/24       64512
[redacted]/24       64512
[redacted]/24       64512
[redacted]/24       64512
[redacted]/24       64513
[redacted]/24       64513
[redacted]/24       64513
[redacted]/24       64514
[redacted]/24       64514
[redacted]/24       64514
[redacted]/24       64514
```

Figure 4.5: ASN Database Example

Next, on the pipeline is the *modify-expert-bot*. This bot will add to the event the field *event_description.text*, according to the AS numbers on the customised ASN database, and the value of this field will be the same as on the field *classification_id* of the YAML files, which is an unique identifier for the target-entities.

If the AS number on the event is not in the customised ASN database or the event does not contain an AS Number the field *event_description.text* will have the value “others”, in order to filter the events that are not related to the target-entities.

The *modify-expert-bot* works using a configuration file in JSON format, where it is specified certain fields of an event, and if the values of those fields match with the given configuration of the *modify-expert-bot*, then the field on the configuration file is added to the event. Figure 4.6 shows an example of the configuration file for the *modify-expert-bot*.

```
{
  “default”: {
    “conficker”: [{
      “malware.name”: “^conficker(ab)?$”
    }, {
      “classification.identifier”: “conficker”
    }],
    “urlzone”: [{
      “malware.name”: “^urlzone2?$”
    }, {
      “classification.identifier”: “urlzone”
    }]
  }
}
```

Figure 4.6: Modify Configuration File Example

For example, if an event contains the field *malware.name* and its value is the name ”conficker”, then the malware name matches with the rule in the configuration file, so the *modify-expert-bot* will add to the event the field *classification.identifier* with the value ”conficker”, as well. This bot allows to enrich the events with information, by changing or adding fields and values to the events, without altering the code of many other bots.

In this project, the configuration file of the *modify-expert-bot* was made with separate sets of rules, each set corresponds to one entity, and there are other two sets of rules for events that are not related to the target-entities. These are events that do not contain the fields *source.asn* and *destination.asn* or if these two fields are present, and their values do not correspond to any AS number of the target entities. If the condition of these two last sets of rules are met, then it is added to the event the field *event_description.text* with the value ”others”. Figure 4.7 shows an example of the modify configuration file with one set of rules for a target-entity and the two sets of rules for events that are not related to the entities.

```
{
“entity-A”: {
  “entity-A-source”: [{
    “source.asn”: “64513”
   }, {
    “event_description.text”: “A”
  }],
  “entity-A-destination”: [{
    “destination.asn”: “64513”
   }, {
    “event_description.text”: “A”
   }]
  },
“no-asn”: {
  “no-source”: [{
    “source.asn”: “”
   }, {
    “event_description.text”: “others”
  }],
  “no-destination”: [{
    “destination.asn”: “”
   }, {
    “event_description.text”: “others”
   }]
  },
“others-asn”: {
  “others-source”: [{
    “source.asn”: “^((?!64[5-9][1-9][2-
9]|65[0-5][0-3][0-4])[0-9]*)$”
   }, {
    “event_description.text”: “others”
  }],
  “others-destination”: [{
    “destination.asn”: “^((?!64[5-9][1-
9][2-9]|65[0-5][0-3][0-4])[0-9]*)$”
   }, {
    “event_description.text”: “others”
   }]
  }
}
```

Figure 4.7: Modify Configuration Rule Sets

Following the *modify-expert-bot* are the *filter-expert-bots*. There is a *filter-bot* for each target-entity and one *filter-bot* for the events that are not related to the target-entities. The events are filtered according to the field *event_description.text*, which was added previously by the *modify-expert-bot* and its corresponding value, which is the unique identifier of each target-entity. The *filter-bots* have a configuration named *filter_action* which has two options, *keep* or *drop* the events, in this case they were configured to *keep* the events. Then the filtered events are sent to the "Pipeline Exit Point" as shown on figure 4.4. The "Exit Point" is composed by different *output-bots*, these could be file texts, at least one per entity, or other type of outputs such as databases or platforms for data analysis.

4.5 Case 2: IP Addresses Second Pipeline

While this project was being developed, IntelMQ suffered several updates which forced a change on the previous pipeline. The major update was in the format of the configuration file for the *modify-expert-bot*. In the the previous version, the *modify-expert-bot* would go through all the sets of rules in a random order, which could cause problems. For example, the random order did not guarantee that the bot would reach all the set of rules, and if it missed one, the events could not be filtered correctly. So, in order to avoid this problem, the configuration file was updated, as well as the bot *modus operandi*. Currently, the bot goes through every rule from the beginning to the end of the file, this way all the rules are read by the bot.

Now, the configuration file does not have rule sets but only specific rules, in the previous version was possible to have a set of rules dedicated only to an entity, now the configuration file has a list of rules. Every rule has a name, and if the event contains a field with a certain value and it matches with the rule in the file, then another field-value pair will be added to the event. This logic is practically equal to the previous version of the *modify-expert-bot*. Figure 4.8 shows the new format of the configuration file for the *modify-expert-bot*.

```
{
  "rulename": "entity-
A_destination_asn"
  "if": {
     "destination.asn": 64513
   },
   "then": {
      "event_description.text": "A"
   }
},
{
  "rulename": "others-asn-source",
  "if": {
     "source.asn": "^((?!64[5-9][1-9][2-
9]|65[0-5][0-3][0-4])[0-9]*)$"
 },
  "then": {
     "event_description.text": "others"
  }
},
{
  "rulename": "others-asn-
destination",
  "if": {
     "destination.asn": "^((?!64[5-9][1-
9][2-9]|65[0-5][0-3][0-4])[0-9]*)$"
   },
 "then": {
   "event_description.text": "others"
  }
},
{
  "rulename": "no-asn-source",
  "if": {
     "source.asn": ""
  },
  "then": {
     "event_description.text": "others"
  }
},
{
  "rulename": "no-asn-destination",
  "if": {
     "destination.asn": ""
  },
  "then": {
     "event_description.text": "others"
  }
}
```


Figure 4.8: New Modify Configuration File

From figure 4.8 is possible to observe that the first rule has the name *entity-A_destination_asn* and **if** in the event there is the field *destination.asn* with the value ”64513”, **then** the field *event_description.text* with the value **A** is added to the event, and so on with the remaining rules.

After this change, the pipeline had to be updated because the modify configuration file lost the rule sets according to each target-entity. The events were no longer filtered correctly. It was necessary to add to the pipeline a *modify-expert-bot* per entity, where each of them had their own configuration file, with rules that belonged to the respective target-entity. Figure 4.9 shows the new pipeline after the update of IntelMQ.

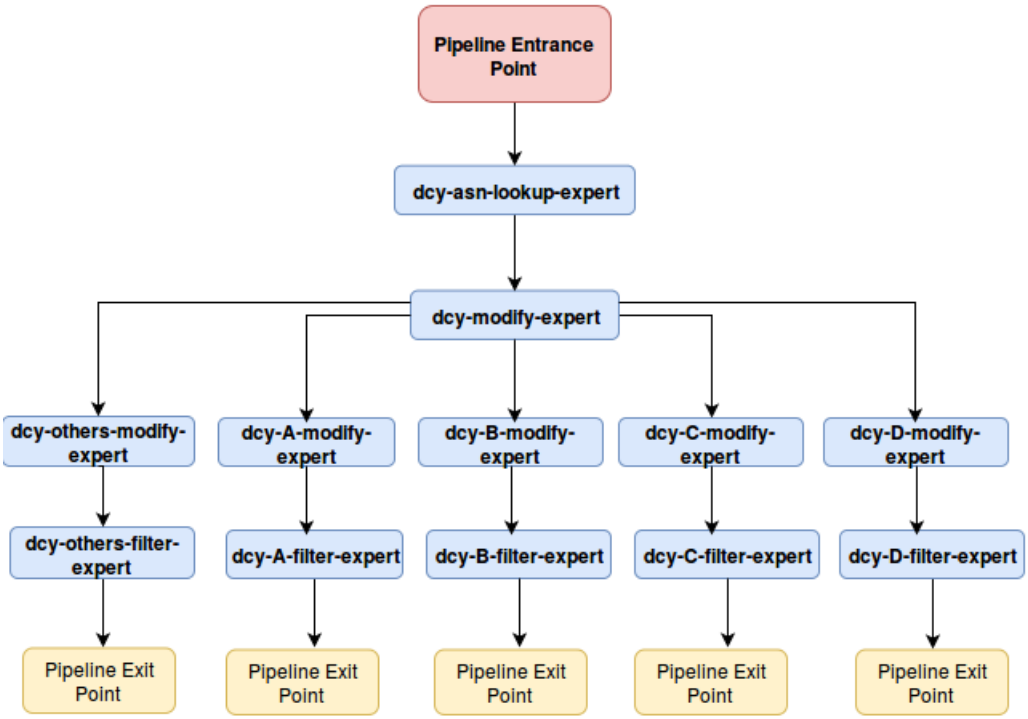

Figure 4.9: New IntelMQ Pipeline

By observing figure 4.9 is possible to verify that the *asn-lookup-bot* was kept in the same position. Then is followed by a general *modify-expert-bot*, this bot has a configuration file that contains all the rules of the target-entities and of the events that are not related to the entities. The general *modify-expert-bot* is necessary in cases where an event that belongs to a target-entity has fields with wrong values. For example, let's suppose there are three target-entities: A, B and C. And as input Cyber Intelligence feed we have figure 4.10.

In figure 4.10 every line of the input feed represents an event. The event in the red rectangle has a *destination.ip* that belongs to **entity-A** and a *source.ip* that is unknown. This event also contains the field *event_description.text*, however this field has the value **C** and it should have the value **A** because this event has an IP address that belongs to **entity-A**. So, when this event reaches the general *modify-expert-bot* the value of the field *event_description.text* will be replaced by the correct one, because this bot has all the rules that belong to the target-entities. And when this event reaches the specific *modify-expert-bot* for **entity-A**, it will reinforce the rule making sure that the field *event_description.text* has the correct value.

Another situation where the general *modify-expert-bot* is useful, is when there are events that involves more than one target-entity, like the event in the green rectangle in figure 4.10. This event needs to be filtered by the *dcy-A-filter-expert* and also by the *dcy-C-filter-expert*, because the *destination.ip* belongs to **entity-A** and the *source.ip* belongs to **entity-C.** The filter-key for all events is the field *event_description.text*, which in this

classification.type	destination.ip	source.abuse_contact	source.ip	event_description.text	destination.asn	source.asn	time.source
blacklist	[redacted]	attack12@foo.com	[redacted]				18/02/2010 00:00
malware	[redacted]	stuff@foo.com	[redacted]				18/02/2010 00:00
blacklist	[redacted]	attack7@foo.com	[redacted]				18/02/2010 00:00
c&c	[redacted]	stuff7@foo.com	[redacted]				18/02/2010 00:00
malware	[redacted]	stuff2@foo.com	[redacted]	C			18/02/2010 00:00
malware	[redacted]	stuff1@foo.com	[redacted]	C			18/02/2010 00:00
blacklist	[redacted]	attack11@foo.com	[redacted]	C			18/02/2010 00:00
spam	[redacted]	stuff11@foo.com	[redacted]	A	64513		18/02/2010 00:00
spam	[redacted]	stuff3@foo.com	[redacted]	A		64513	18/02/2010 00:00
malware	[redacted]	attack2@foo.com	[redacted]	A	78569		18/02/2010 00:00
malware	[redacted]	attack5@foo.com	[redacted]	C		1236547	18/02/2010 00:00
malware	[redacted]	attack@foo.com	[redacted]	B	64514		18/02/2010 00:00
blacklist	[redacted]	attack8@foo.com	[redacted]	B		64512	18/02/2010 00:00
blacklist	[redacted]	attack13@foo.com	[redacted]	C	64512		18/02/2010 00:00
blacklist	[redacted]	attack10@foo.com	[redacted]	B	123456	64512	18/02/2010 00:00
malware	[redacted]	attack3@foo.com	[redacted]	A	45456	46565	18/02/2010 00:00
malware	[redacted]	pedroreis@things.com	[redacted]	B	64514	64514	18/02/2010 00:00
c&c	[redacted]	stuff6@foo.com	[redacted]	C	12364	64513	18/02/2010 00:00
c&c	[redacted]	stuff8@foo.com	[redacted]	A	64512	64514	18/02/2010 00:00
malware	[redacted]	attack1@foo.com	[redacted]	B	64513	64512	18/02/2010 00:00
blacklist	[redacted]	attack9@foo.com	[redacted]			123654	18/02/2010 00:00
malware	[redacted]	attack4@foo.com	[redacted]		47896		18/02/2010 00:00
blacklist	[redacted]	stuff9@foo.com	[redacted]				18/02/2010 00:00
blacklist	[redacted]	stuff10@foo.com	[redacted]				18/02/2010 00:00
c&c	[redacted]	stuff5@foo.com	[redacted]				18/02/2010 00:00

Keys:
Entity-A
Entity-B
Entity-C

Figure 4.10: Example of a Cyber Intelligence Feed

case has the value **B**, so this value must be corrected for both entities. The general *modify-expert-bot* will correct this value for **entity-A**, because the rules in its configuration are in alphabetic order, then when the event reaches the *dcy-A-modify-expert* the rules will be reinforced to make sure that all the fields have the correct values for **entity-A**. But when the event reaches the *dcy-C-modify-expert*, the field *event_description.text* will still have the value **A**, because of the general *modify-expert-bot*. So, the *dcy-C-modify-expert* will correct the value to **C**, because this event also belongs to **entity-C**.

In the pipeline, after the *modify-bots* there are the *filter-bots*, one per entity and another for events that are not related to the target-entities. As mentioned previously, the filter-key used by all the *filter-bots* is the field *event_description.text*, for example the value for this field for entities A, B and C are exactly **A**, **B** and **C**. All the entity filters are configured to *keep* the events, in order to send them to a file or to another type of output. However, there is a difference between the filters of the target-entities and the filter for others events.

It is very difficult to identify all the events that are not related to the target-entities, because it is not known what characterises them. But, what is known, is what characterises every single target-entity. Hereupon, it was added to the *dcy-others-modify-expert-bot* configuration file, rules that belong to the target-entities, but instead of giving the values A, B or C according to the respective target-entity, it was added the field *event_description.target* with the value **entities**, because those rules belong to the them. Therefore, the *filter-bot* for the events that are not related to the target-entities is configured to *drop* the events that are related to the target-entities, according to the key

event_description.target and the value **entities**, in this way the bot will filter the events that do not belong to the target-entities.

```
{
 "rulename": "special-asn-source",
 "if": {
  "source.asn": "(6451[2-9]|...|6553[0-4])"
 },
 "then": {
  "event_description.target": "entities"
 }
},
{
 "rulename": "special-asn-destination",
 "if": {
  "destination.asn": "(6451[2-9]|...|6553[0-4])"
 },
 "then": {
  "event_description.target": "entities"
 }
}
```

Figure 4.11: Example of Rules in the dcy-others-modify-expert Configuration File

The above figure represents two rules for the AS numbers of the target-entities in the configuration file of the *dcy-others-modify-expert*. Both rules contain a regular expression, a sequence of symbols and characters expressing a string or pattern to be searched for, within a longer piece of text. In this case it represents the range 64512 to 65534 inclusive. If an AS number is found within this range, then is added to event the field *event_description.target* with the value **entities**.

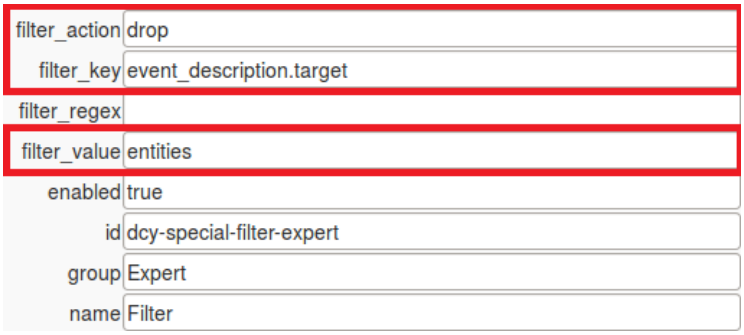

Figure 4.12: Configuration Example for *dcy-others-filter-expert*

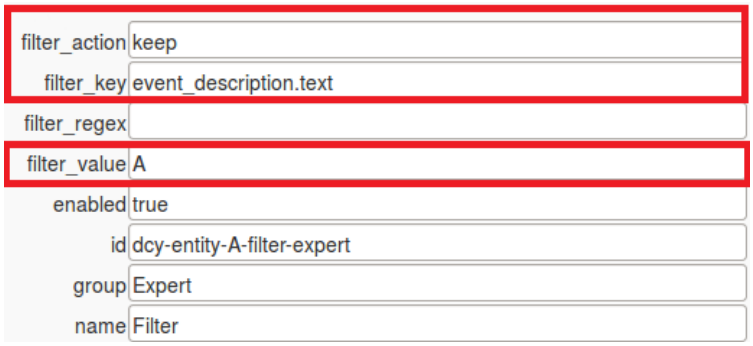

Figure 4.13: Configuration Example for *entity-filter-expert*

Figures 4.12 and 4.13 represent the configuration for the *filter-bots*, the *filter_action* used, *keep* or *drop*, the *filter_key* and the *filter_value* are in the red rectangles.

4.6 Case 3: FQDNs and URLs

Following the filtering of IP addresses of the target-entities, it was necessary to filter events that contained FQDNs and URLs of the entities. For this, it was not necessary to change the previous pipeline, which means that was not required to add more bots to it.

The only change made was in the *modify-expert-bots*. It was added to the *modify-expert-bots* configuration files, rules involving the FQDNs and the URLs of the target-entities. For example, if an event contained an URL that belonged to **entity-A**, then it was added to the event the field *event_description.text* with the value **A**. All the rules were duplicated, because the FQDNs and URLs in the events might come as source or destination, so there are two rules for each FQDN and URL, as shown in figure 4.14.

```
{
 "rulename": "entity-A_source_url_1",
 "if": {
  "source.url": "http://www.entityA.com"
 },
 "then": {
  "event_description.text": "A"
 }
},
{
 "rulename": "entity-A_destination_url_1",
 "if": {
  "destination.url": "http://www.entityA.com"
 },
 "then": {
  "event_description.text": "A"
 }
}
```

Figure 4.14: Example of Rules for URLs

To filter the events that are not related to the target-entities, it was applied the same logic as for the IP addresses. The configuration file of the *dcy-others-modify-expert*, contains rules related to the FQDNs and URLs, but instead of adding the corresponding value associated to the entity, it adds the value **entities** this way the *dcy-others-filter-expert* will recognise which events are related to the target-entitites and will *drop* them, leaving only the ”others” events.

```
{
 "rulename": "entity-A_source_fqdn_1",
 "if": {
  "source.fqdn": "entityA.com"
 },
 "then": {
  "event_description.target": "entities"
 }
},
{
 "rulename": "entity-A_destination_fqdn_1",
 "if": {
  "destination.fqdn": "entityA.com"
 },
 "then": {
  "event_description.target": "entities"
 }
}
```

Figure 4.15: Example of Rules of the *dcy-others-modify-expert* Configuration File

4.7 Maltego to IntelMQ Architecture

This section describes the overall architecture between Maltego and IntelMQ and the expected outputs from it.

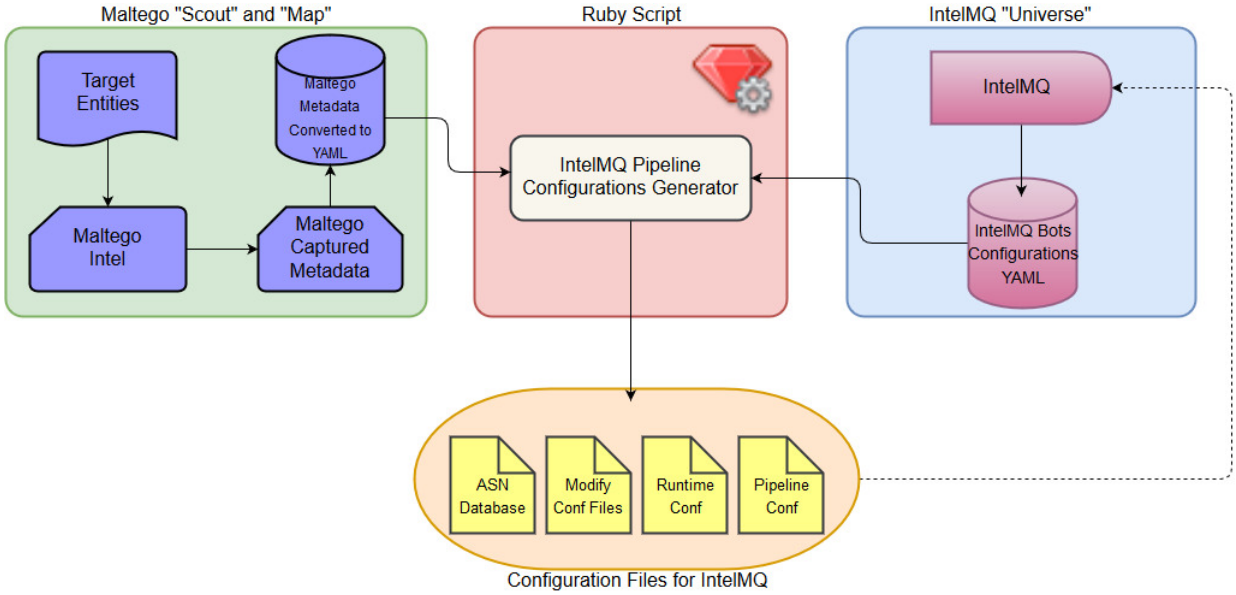

Figure 4.16: IntelMQ Configuration Manager Architecture

Figure 4.16 shows an overview of the architecture between Maltego, IntelMQ and the Ruby program which generates the configurations for IntelMQ. At the left side of the figure there are the target-entities, they must be known and defined before the progress of the project. After this, they were mapped and scouted by Maltego and it was found their most relevant attributes. The captured information was then manually transformed into YAML format. Each entity has their own YAML file with their IP addresses, URLs and FQDNs. These YAML files served as input for the Ruby program that generates the configuration files for IntelMQ. The Ruby program also needs as input an extra YAML file, that contains the destination or origin file paths for the creation of the configuration files and it also contains bots configurations. The Ruby program then returns several files as output, namely: ASN database, various modify configuration files and the files *runtime.conf* and *pipeline.conf*. All the files are sent to the correct directory, this way IntelMQ can read them automatically.

4.8 Conclusion

This chapter described how the solution to filter the events related to the target-entities was found. It explained the requirements that the Ruby program must reach. How the public information on the target-entities was collected by Maltego and what set of transforms were used to reach this goal. It also explains how the metadata from Maltego

was transformed into YAML format. Then it is described the evolution of the IntelMQ pipelines for the correct filtering of events, this also includes a detailed description of the configuration files and bots configurations. Finally, it was explained a general overview of the architecture between Maltego, IntelMQ and the Ruby Program.

The following chapter describes the implementation of the Ruby program, its methods and functions that are responsible for the generation of the configuration files, bots configurations and the generation of the pipeline, which filters the events accordingly.

Chapter 5

Implementation

This chapter describes the Ruby program that was developed for this project, the program was named as *"intelmq_configurations_generator.rb"*. Its main goal was to generate all the configuration files for the *modify-expert-bots*, generate the ASN database, the configurations for the the *filter-expert-bots* and the files *runtime.conf* and *pipeline.conf*, which contain all the bots configurations and their connections in the pipeline.

The program uses the YAML files that contain information of the target-entities to generate all the configurations files, these files will contain the rules for the correct filtering of the events. The program also uses an extra YAML file, that contains the paths from where the configuration files must be created, the bots names and configurations and also some commands to be executed by IntelMQ.

The first section of this chapter describes the building blocks of the Ruby script *"intelmq_configurations_generator.rb"*. The second section describes IntelMQ configuration YAML file, its content and its purpose for the Ruby program. The third section starts describing the first part of the code, which is the generation of entities objects with their attributes from the YAML files. The fourth section explains how the *asn-lookup-expert* and its database was generated, how the configurations files for the *modify-expert-bots* were created and also how the configurations of the *filter-expert* were made. The following sections explain how the files *runtime.conf* and *pipeline.conf* were generated. Section 5.7 describes the command line switch that was created to execute the different functions of the Ruby program in a terminal. The last section of this chapter summarises all the outputs, that the *"intelmq_configurations_generator.rb"* generates.

5.1 IntelMQ Configurations Generator Overview

The purpose of this section is to describe the overall functioning of the the Ruby program, developed for this project. Figure 5.1 shows an overview of the different sections of the program.

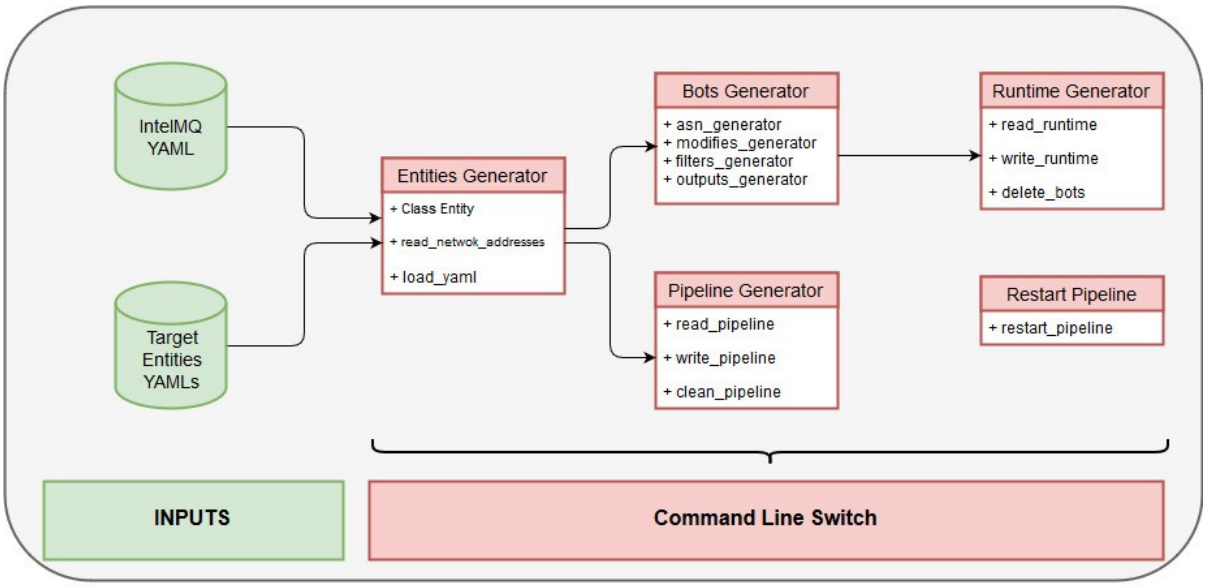

Figure 5.1: IntelMQ Configurations Generator Overview

On the left side of the figure are the input files for the program, these are the target-entity YAMLs, and the IntelMQ YAML file, which is described in detail in section 5.2. The first methods of the Ruby script are responsible for the reading of the YAMLs of the target-entities, and create entities objects, this way will be possible to use their information throughout the whole program, without any losses.

The next section of the script is called ”Bots Generator”, it generates the ASN database, the configurations files for the *modify-expert-bots* and the configurations for the *filter-experts-bots* and *output-bots*. Following is the ”Runtime Generator”, and as its name indicates it generates the file *runtime.conf*, this file contains all the bots configurations. To generate *runtime.conf*, the methods in ”Runtime Generator” use functions from the section ”Bots Generator”.

Then there is the section ”Pipeline Generator”, this is where the file *pipeline.conf* is generated, this file contains the connections between the bots.

The script also contains a method responsible of the restart of the IntelMQ Pipeline, this method does not use any information of the previous methods. It executes commands that belong to the framework Intelmqctl, which was explained in chapter 3.

Finally, there is the ”Command Line Switch” which is responsible for executing the methods through the command line.

Throughout this chapter it will be explained in detail every section of the Ruby program *”intelmq_configurations_generator.rb”*.

5.2 IntelMQ Configuration File

This section describes the YAML file with configurations for IntelMQ, which served as input for the *intelmq_configurations_generator.rb*, in order to generate all the necessary configuration files for the project. The file contains all the static information necessary for the Ruby program and it is composed by field names and their respective values, the program reaches the values by using the field names.

This file is organised in different sections, the first part contains the bots names, the prefix to be used in the bots that were generated by the Ruby program, which works as an identifier for these bots. In this case, the prefix used was ”dcy”, because these are the initials of the department where the project was developed - Direção de Cyber Security and Privacy (DCY). All the bots generated by the Ruby program will have the prefix ”dcy”. This section also contains the entrance point for the pipeline generated by the program. This way the generated pipeline can be attached to any bot that was already in IntelMQ and which is specified in this YAML file. Figure 5.2 shows the bots names and tags used in this project, as well as, the entrance point for the pipeline.

```
#bots tags and names
bots_tag: mtf #prefix to be used to identify the bots that were generated by the Ruby scripts
asn_name: asn-lookup-expert #name to be used for bot identification
modify_name: modify-expert #name to be used for bot identification
modify_url_fqdn: url-fqdn
filter_name: filter-expert #name to be used for bot identification
deduplicator_name: deduplicator-expert #name to used for bot identification
output_name: redis-output #name to be used for bot identification
output_file: file-output #name to be used for bot identification
others_tag: others #tag to be used to filter 'other' events
special_tag: special #tag to be used by special filter bot
entities_tag: entities #value to be used by special filter bot
queue_name: queue  #word to add to bots queues
global_name: global #tag to be used on global output bots
entrance_point: taxonomy-expert #entrance point from pipeline (bot name)
```

Figure 5.2: First Section of the IntelMQ Configuration YAML File

The next section is composed by commands to run the program Intelmqctl, which was described on chapter 4. These commands are necessary in cases where the pipeline that was generated by *”intelmq_configurations_generator.rb”* is no longer necessary, and it is necessary to remove the pipeline from IntelMQ, the bots must be stopped before deleting the pipeline. Or if for some reason the bots crash or block, the best way to solve this issue is to restart them, once again is used the approriate command to do so. The following figure demonstrates examples of the commands to execute Intelmqctl.

```
#intelmqctl commands
bot_restart_command: intelmqctl restart
bot_stop_command: intelmqctl stop
bot_clear_queue_command: intelmqctl clear
get_bots_command: intelmqctl --type json list bots
```

Figure 5.3: Intelmqctl Commands

The following section (figure 5.4) contains all the static parts to be used in the rule names for the *modify-expert-bots* configuration files. These rules names contain the source / destination AS numbers and source / destination FQDN and URL.

```
#rule names for modify conf file bot
source_asn: _source_asn
destination_asn: _destination_asn
source_fqdn: _source_fqdn
destination_fqdn: _destination_fqdn
source_url: _source_url
destination_url: _destination_url
```

Figure 5.4: Modify Rule Names

Following the previous section, there are the paths of the files *runtime.conf* and *pipeline.conf* and the path to the directory which contains all the target-entity YAML files. Next, there are the paths of the files to be created by the Ruby program, it contains the paths for the *modify-expert-bots* configuration files, the ASN database and backups of the *runtime.conf* and *pipeline.conf* files, as figure 5.5 shows.

```
#paths of files to be created
asn_database: /opt/intelmq/var/lib/bots/asn_lookup/out.dat
modify_conf_file: /opt/intelmq/var/lib/bots/modify/modify.conf
modify_others: /opt/intelmq/var/lib/bots/modify/others_modify.conf
modify_entities_path: /opt/intelmq/var/lib/bots/modify/
modify_template: /opt/intelmq/ruby_intelmq /modify_template.json
runtime_backup: /opt/intelmq/etc/examples/runtime_original_backup.conf
pipeline_backup: /opt/intelmq/etc/examples/pipeline_original_backup.conf
runtime_new: /opt/intelmq/etc/runtime.conf
pipeline_new: /opt/intelmq/etc/pipeline.conf
runtime_v2_backup: /opt/intelmq/etc/examples/runtime.conf
pipeline_v2_backup: /opt/intelmq/etc/examples/pipeline.conf
global_output: /opt/intelmq/var/lib/bots/file-output/global_output.txt
```

Figure 5.5: Paths for the Configuration Files

Finally, figure 5.6 presents the last section of this file, it is composed by the configurations of the bots used in the pipeline, such as the *asn-lookup-expert*, the *modify-experts*

and the *filter-experts* bots. The bots configuration is kept in this file, because most of them contain static information, while in the Ruby program there should be only dynamic data.

```
asn_lookup:
  parameters:
    database:/opt/intelmq/asn_lookup/out.dat
  group: Expert
  name: ASN Lookup
  module: intelmq.bots.experts.asn_lookup.expert
  description: modified by mfelix

modify:
  parameters:
    configuration_path:
/opt/intelmq/modify/modify.conf
  name: Modify
  group: Expert
  module: intelmq.bots.experts.modify.expert
description: modified by mfelix
```

```
special_filter:
  parameters:
    filter_action: drop
    filter_key: event_description.target
    filter_regex: ""
    filter_value: entities
  group: Expert
  name: Filter
  module: intelmq.bots.experts.filter.expert
  description: modified by mfelix

filter:
  parameters:
    filter_action: keep
    filter_key: event_description.text
    filter_regex: ""
    filter_value: ""
  group: Expert
  name: Filter
  module: intelmq.bots.experts.filter.expert
description: modified by mfelix
```

Figure 5.6: Bots Configurations

5.3 Entities Generator

This section describes how the **entity** objects were developed in the Ruby program.

An object-oriented program involves classes and objects. A class is the blueprint from which individual objects are created. In object-oriented terms, is said that an **entity** is an instance of the class of objects known as **entities** [46].

The part of the code which generates the **entity** objects is called Entities Generator and is by a class named *Entity* and two methods named *read_network_addresses* and *load_yaml*. The class *Entity* declares and initialises all the attributes for an **entity** object, these are the same as in the target-entities YAML files. The method *read_network_addresses* saves the different sections of IP addresses or network names into separate arrays from the entities YAML files. However, if a target-entity does not have IP addresses or only contains one of the three types of network addresses (single IP, range or subnet) or network names (URLs, FQDNs or domains), it ignores the missing or non-existing network addresses / names.

The previous method is followed by *load_yaml* method, which is the most complex part of the Entities Generator. The first part of *load_yaml* method is to find all the entity

YAML files in the given directory in the *intelmq_conf* (shown in the previous section, this file contains all the paths of the YAML files). The method searches in the given directory all the files with the extension ".yaml". Then, each file is opened and it id verified if the files do not contain any errors, if they are in the correct YAML format and if they have the correct syntax. After opening the files, all the content from the network addresses and network names is saved into arrays through the method *read_network_addresses*, this happens if they are present in the YAML files, otherwise it will ignore the absent fields. The method *load_yaml*, also adds the AS numbers dynamically to each entity object, this way is avoided repeated AS numbers and is ensured they are in the range 64512 to 65534. To build the entity objects, the function also uses a map / dictionary of fields from the entities YAML files. This map helps the main function to reach the desired fields in the YAML files.

5.4 Bots Generator

This section describes how the Ruby program generates the ASN database used by the *asn-lookup-expert*, the configuration files for the *modify-expert-bots* and the configurations for the *filter-expert-bots* used in the pipeline. Therefore, this section is divided into three subsections, each one belongs to one type of expert bots. The first subection is focused in the creation of the AS database for the *asn-lookup-expert*, the second subsection describes the code for the creation of the configuration files for each *modify-expert* and the third subsection explains the development of the configurations for the *filter-experts*.

5.4.1 ASN-Lookup-Expert / ASN Database

This subsection, as mentioned previously, describes how the ASN database is generated by the Ruby program.

The method which originates the ASN database is called *write_asn_database*. The first part of this method consists on the writing of the header of the database. The header for the database is in the *intelmq_conf.yaml*. An enhancement for this header it would be if it was built dynamically everytime the database was created or updated, because the header should contain the date of creation and the total number of IPv4 and / or IPv6 addresses in the database. This was not made during the project, because the main goal of this database is having the correct IP addresses and AS numbers, the header is just a reference for its content.

After writing the header to a file, the next part of the method *write_asn_database* will get the single IP addresses, ranges and subnets from the entities objects, generated by the method *load_yaml*, described on the previous section. Each of the network addresses type will be merged into the CIDR format and then saved into an array. The merging process allows to reduce the number of IP addresses, for example, a target-entity might have

several single IP addresses which belong to the same range, by merging them is possible to reduce the quantity of IP addresses. After merging each network type separatly, all the network types were merged once again (single IP addresses, ranges and subnets), on this second merge they were already in the CIDR format, reducing again the number of IP addresses to be written in the database. After reducing the quantity of IP addresses by merging them, they were written to the database with the corresponding AS number, according to the target-entity. They were written as pairs of IP addresses / AS numbers, which means that the database has two columns, one for IP addresses and another for the corresponding AS numbers, as shown in chapter 4 in figure 4.5.

The name for the database and the directory where it should be written and saved is specified in the *intelmq_conf.yaml*.

5.4.2 Modify-Expert-Bots

Following the method that generates the ASN database, there are the methods which generate the configuration files for the *modify-expert-bots*. To build the rules with these methods, it was used a template file, named *modify_template.json* with all the static field names for the rules, as shown in figure 5.7.

```
{
  "rulename": "fqdn-source",
  "if": {
     "source.fqdn": ""
  },
  "then": {
    "event_description.text": ""
  }
},
{
 "rulename": "fqdn-destination",
 "if": {
    "destination.fqdn": ""
  },
  "then": {
    "event_description.text": ""
  }
}
```

Figure 5.7: Rule Example from *modify_template.json*

The first method is called *generate_modify_url_fqdn*, this method constructes the rules for the URLs and FQDNs of the target-entities and they are separated into four sets of rules: *destination_fqdn*, *source_fqdn*, *destination_url* and *source_url*. For each set, the method gets the template rules that are in the modify template file, afterwards it fills each rule with data from the target-entities.

```
fqdn_source = Hash.new
fqdn_destination = Hash.new
url_source = Hash.new
url_destination = Hash.new

template =  INTELMQ_CONF['intelmq_conf_file']['modify_template']

modify_template = File.read(template)
template_hash = JSON.parse(modify_template)

template_hash.each do |rule|
  if rule['rulename'].match('fqdn-source')
    fqdn_source = Marshal.load(Marshal.dump(rule))
  end
  if rule['rulename'].match('fqdn-destination')
    fqdn_destination = Marshal.load(Marshal.dump(rule))
  end
  if rule['rulename'].match('url-source')
    url_source = Marshal.load(Marshal.dump(rule))
  end
  if rule['rulename'].match('url-destination')
    url_destination = Marshal.load(Marshal.dump(rule))
  end
end
```

Figure 5.8: Template Rules for Each Set of Rules for the Target Entities

```
array_rules = Array.new
ENTITIES.each do |entity|
  id = entity.classification_id
  fqdn = entity.public_fqdn
  url = entity.public_url
  name = entity.name

  source_f = 0
  dest_f = 0
  fqdn.each do |item|
    fqdn_source['rulename'] = name + source_fqdn + '_' + "#{source_f+=1}"
    fqdn_source['if']['source.fqdn'] = item
    fqdn_source['then']['event_description.text'] = id
    array_rules << Marshal.load(Marshal.dump(fqdn_source))

    fqdn_destination['rulename'] = name + destination_fqdn + '_' + "#{dest_f+=1}"
    fqdn_destination['if']['destination.fqdn'] = item
    fqdn_destination['then']['event_description.text'] = id
    array_rules << Marshal.load(Marshal.dump(fqdn_destination))
  end
```

Figure 5.9: Filling of the Rule Sets for the FQDNs

In figure 5.8 is possible to observe the creation of the four sets of rules, then the file *modify_template.json* is read by the method and the static names of the rules were added to

each rule set. While adding the static names to the rule sets is used a Ruby Module named *"Marshal"*. This module converts collections of Ruby objects into a byte stream, allowing them to be stored outside the currently active script. This allows to create a shallow copy of the Ruby object, being possible to alter the copied object without changing the original [1]. Without this module, all the rule sets would have the same values, which would cause problems during the filtering of the events.

The next part of the method is responsible for the filling of the rules with the URLs and FQDNs of the target-entities, as shown in figure 5.9. For this it is used the *classification_id*, the entity name and the URL and FQDN of each target-entity. It was also added a counter to each rule name, in order to avoid repetition in the rule names, for example: *entity-A_source_url_1"*. All the rules are numerated, making each rule name different from the previous. Then the rules are all saved into an array.

The next method is called *generate_special_modify_url_fqdn* and is very similar to the method *generate_modify_url_fqdn*. The difference is in the field to be added in the event and its value. In the previous method the rule contained the field **event_description.text** and value added to it was the *classification_id* of the corresponding target-entity, while in this method the rule contains the field **event_description.target** and the value added to it is the tag **entities**. As mentioned previously, these rules are meant for the *dcy-others-modify-expert*, in order to filter the events that are not related to the target-entities.

```
ENTITIES.each do |entity|
  fqdn = entity.public_fqdn
  url = entity.public_url
  name = entity.name

  source_f = 0
  dest_f = 0
  fqdn.each do |item|
    fqdn_source['rulename'] = name + source_fqdn + '_' + "#{source_f+=1}"
    fqdn_source['if']['source.fqdn'] = item
    fqdn_source['then']['event_description.target'] = entities_tag
    array_rules << Marshal.load(Marshal.dump(fqdn_source))

    fqdn_destination['rulename'] = name + destination_fqdn + '_' + "#{dest_f+=1}"
    fqdn_destination['if']['destination.fqdn'] = item
    fqdn_destination['then']['event_description.target'] = entities_tag
    array_rules << Marshal.load(Marshal.dump(fqdn_destination))
```

Figure 5.10: Special Rules Filling

Figure 5.10 shows a fragment of code which demonstrates the field name to be added to the event and its value in the method *generate_special_modify_url_fqdn*. Once again, all the rules were saved into an array.

The following method's name is *generate_modify_entities*. This method writes the rules that belong to the target-entities into separate files. In other words, it generates the

modify configuration files for each *modify-expert-bot* of each target-entity. The content of this method is similar to the previous two methods, however it also adds rules related to AS numbers of the target-entities and instead of saving the rules into an array, they are written directly to the corresponding files.

```
  asn_source['rulename'] = entity.name + source_asn
  asn_source['if']['source.asn'] = asn
  asn_source['then']['event_description.text'] = id
  array_rules << Marshal.load(Marshal.dump(asn_source))

  asn_destination['rulename'] = entity.name + destination_asn
  asn_destination['if']['destination.asn'] = asn
  asn_destination['then']['event_description.text'] = id
  array_rules << Marshal.load(Marshal.dump(asn_destination))
end

#writes rules to each entity modify conf file
ENTITIES.each do |entity|
  File.open(modify_path + entity.name + '-' + modify_file_name, 'w') do |f|
  matching_rules = array_rules.select{ |rule| rule['rulename'].match(entity.name) }
  f.write(JSON.pretty_generate(matching_rules))
  f.close
end
```

Figure 5.11: Entities Rules Writing

```
array_final = Array.new
array_final << special_asn_source
array_final << special_asn_dest
array_final << others_asn_source
array_final << others_asn_dest
array_final << no_asn_source
array_final << no_asn_dest

#gets array_rules from generate_special_modify_url_fqdn
special_url_fqdn = generate_special_modify_url_fqdn(conf_file)

#adds special rules to array_final
special_url_fqdn.each do |rule|
  array_final << rule
end

#writes array_final to modify others events conf file
File.open(INTELMQ_CONF['intelmq_conf_file']['modify_others'], 'w') do |f|
  f.write(JSON.pretty_generate(array_final))
  f.close
end
```

Figure 5.12: Others Rules Writing

By observing figure 5.11, is possible verify the code part that creates the rules for the

AS numbers of the target-entities and also how the rules are written into separate files, each file corresponding to one target-entity. The path to where the files must be written and their names are in the *intelmq_conf.yaml*, which was described in the second section of this chapter.

The next method is responsible for the writing of the rules of the events that are not related to the target-entities. The name of the method is *generate_others_modify_file*. Its structure is similar to the previous methods. However, the set of rules is different, this file contains rules for events that do not contain AS numbers, or if they contain AS numbers they are different from the AS numbers of the target-entities. All the rules are saved into an array. This method calls the function *generate_special_modify_url_fqdn* and writes its output to the file along with the rules of the different AS numbers.

In figure 5.12 is possible to observe the rule sets for the different types of AS numbers and the rule sets for events that do not contain AS numbers being saved into the array. Then the method *generate_special_modify_url_fqdn* is called and each rule of this method is also saved in the array. Finally, the file for these rules is generated and all the rules in the array are written in it.

The following method, *generate_modify_content*, generates the content for the configuration file of the general *modify-expert*, which is the bot next to the *asn-lookup-expert-bot* in the pipeline. As mentioned previously, the general *modify-expert* contains all the rules that belong to the target-entities and also the rules that are not related to them, the content of the configuration files of the *modify-experts* of each target-entity are written to this general file, as well as the rules of the ASN numbers that do not belong to the target-entities. Following this method is the function that generates the general configuration file by writing the output of the *generate_modify_content*. It was created a method only for the writing of the file, because this way is given the possibility to name the file differently from what is specified in the *intelmq_conf.yaml*.

Figure 5.13 presents the code for the writing of the general *modify-expert* configuration file. If a name for the file to be created is not given, then it is used the specified name in the *intelmq_conf.yaml*, then it calls the method *generate_modify_content* and writes its output into the file. Otherwise, the file will be created with the given name and it follows the same logic as the first part of the code, by calling the method *generate_modify_content* and writing its output.

The last method related to this section is the *generate_modify_bots_entities*. This method is responsible for the generation of the configurations for the *modify-expert-bots* for each target-entity. The general *modify-expert-bot* is generated by a method in the Runtime section of this chapter.

```
def generate_modify_conf_file(file_name,conf_file)
  #input: modify file name to be created and intelmq conf yaml
  #output: modify configuration file with the input name
  # generates modify configuration file
  # file format: json file with hashes on the following format:  [ { {}, {} } ]

  if file_name.to_s.empty?
    File.open(INTELMQ_CONF['intelmq_conf_file']['modify_conf_file'], 'w') do |f|
      f.write(JSON.pretty_generate(generate_modify_content(conf_file)))
      f.close
    end
  else
    File.open(file_name, 'w') do |f|
      f.write(JSON.pretty_generate(generate_modify_content(conf_file)))
      f.close
    end
  end
end
```

Figure 5.13: Method to Generate the General Modify Configuration File

```
def generate_modify_bots_entities(conf_file)
  # input: intelmq conf file
  # output: modify bots with their configurations
  # generates modify bots for each entity

  modify = INTELMQ_CONF['intelmq_conf_file']['modify']
  modify_name = INTELMQ_CONF['intelmq_conf_file']['modify_name']
  tag = INTELMQ_CONF['intelmq_conf_file']['bots_tag']
  modify_path = INTELMQ_CONF['intelmq_conf_file']['modify_entities_path']
  modify_file_name = INTELMQ_CONF['intelmq_conf_file']['modify_entities_file_name']

  hash_modify = Hash.new

  ENTITIES.each do |entity|
    modify['parameters']['configuration_path'] = modify_path + entity.name + '-' +
modify_file_name
    modify_aux = Marshal.load(Marshal.dump(modify))
    hash_modify["#{tag + '-' + entity.name + '-' + modify_name}"] = modify_aux
  end

  return hash_modify
end
```

Figure 5.14: Method That Generates the *Modify-Experts-Bots* for Each Target-Entity

The above figure shows the method *generate_modify_bots_entities*. The first part of the method gets all the necessary information from the file *intelmq_conf.yaml*, the general configuration, bot name and name of the configuration file for the *modify-expert-bot*, the path where the configuration files must be written and the tag which identifies each bot

generated by the Ruby program. The following part, attributes dynamically the configuration file name according to an entity. For example, if there are ten target-entities, then it will be generated ten *modify-expert-bots*, each one with a configuration file corresponding to the respective entity. Then each bot configuration is saved into an hash, under a specific name related to the target-entity. These configurations will be then written into the *runtime.conf* file, which will be explained in more detail in section 5.5 of this chapter.

5.4.3 Filter-Expert-Bots

Following the last method from the previous section, there is the method *generate_filter_bots*. The first part of this function gets all the necessary information for the construction of the configurations for the *filter-bots*. These are the general configuration set, name and tag for the bots identification. Then the attributes of each target-entity are added to the configurations dynamically, in order to have a configuration per entity. Finally, all the configurations are added to an hash, and once again, the hash with the configurations will be used by the method which reproduces the *runtime.conf* file.

5.5 Runtime Generator

This section describes the methods which are responsible for the reproduction of the *runtime.conf* file. This part of the code is composed by three methods, the first method is responsible of reading the ”original” *runtime.conf* file, this file already exists on IntelMQ and it contains the configurations of the bots that are in use in IntelMQ. After reading the file, the method converts its content into an hash and it is added to this hash the configurations of the bots that are responsible for the correct filtering of Cyber Intelligence events according to the attributes of the target-entities. The second method of this section writes the hash generated by the previous method into a file, and it also creates two backup files, the first backup is of the original configuration and the second is a copy of the new configuration, both files are created with a timestamp, in order to maintain a track of when they were created. The third method is responsible for the elimination of the bots that were added to *runtime.conf* to filter the events that belong to the target-entities. This last method allows to clear up the pipeline, returning it to its original state, without the bots that were generated by the Ruby program. This is useful in cases where it is necessary to change the target-entities, so the bots are removed from the pipeline and then the script is executed again but with new entities, and new bots with new configurations.

As mentioned previously, the first method of this section reads the original *runtime.conf* file and returns an hash with the configurations that were already in the file and also the new configurations of the bots that will filter the Cyber Intelligence events related to the target-entities. This method is called *read_runtime* and figure 5.15 and shows the first part of this method. The first part of the method is responsible of the reading of the file *run-*

time.conf and transform it into an hash. Then it is collected all the necessary information from the *intelmq_conf.yaml* file to build the bot names and configurations which filter the events of the target-entities. It also calls functions that were described previously in this chapter, such as the *generate_modify_bots_entities* and *generate_filter_bots*. The output of these methods is added to the hash, completing it with all the required configurations.

```
def read_runtime(conf_file)
  # input: intelmq yaml conf file
  # output: a hash which contains the new experts bots: asn, modify and filters
  # reads runtime.conf and intelmq_conf.yaml files and adds expert bots to a hash

  file = File.read(INTELMQ_CONF['intelmq_conf_file']['runtime_conf_path'])
  result_hash = JSON.parse(file)
```

Figure 5.15: Reading of *runtime.conf*

The following method is named *write_runtime*, and it is responsible of writting the output of the *read_runtime* into the *runtime.conf* file. It also creates a backup of the original configurations, adding a timestamp to the file name and another backup file containing the content of the new generated file. These backup files are useful for debug purposes and to keep a record of all the changes made to the file (before and after).

The last method of this section deletes all the bots which contain the tag ”dcy” from the *runtime.conf* file, in other words, it deletes the bots that were generated by the Ruby program. Before deleting the bots, this method stops them first in IntelMQ, this step avoids the loss of information before deleting the bots. After stopping the bots, the method executes the deletion part, and writes a new clean *runtime.conf* file.

5.6 Pipeline Generator

After writting the bots configurations is necessary to create the links between them, in order to have a logical pipeline. This section describes the methods responsible for the creation of the connections between the bots that were generated by the Ruby program, but also the connection of these new bots to the existing pipeline on IntelMQ. Similarly, to the previous section, the Pipeline Generator is also composed by three methods, *read_pipeline*, *write_pipeline* and *clean_pipeline*. The first method reads the original pipeline file, it also converts its content into an hash, aftewards its added to it the queues of bots creating the connections between them. The second method writes the output hash of the previous method, following the same logic as the writing method for the *runtime.conf* file, a backup of the original *pipeline.conf* file is written, as well as a copy of the new generated *pipeline.conf* file. The last method removes the queues that belong to the bots that were generated by the Ruby program, returning the file *pipeline.conf* to its original state.

The first part of the method *read_pipeline* reads the existing *pipeline.conf* file and saves its content to an hash, and as other methods described previously, it also fetches the required information to build the new pipeline from the *intelmq_conf.yaml* file. In this case, it fetches information about the bots names, the tag to be used also in their names and the entrance point. The entrance point is the bot name where the new pipeline is going to be attached to. This method also generates an hash per bot, each hash contains the source-queue and destination-queues for every bot, only the *output-bots* do not contain destination-queues, as they are the end of the pipeline. The source and destination queues are the bot names where the events must pass through, and the queues create the connections between the bots. The next part of the method verifies if already exists a destination-queue in the entrance point, if exists, it adds the first bot of the new pipeline, which is the *asn-expert-bot*, otherwise it adds the destination-queue to the entrance point and again it creates the *asn-expert-bot*, as seen in figure 5.16.

```
if pipeline_hash.keys.any?{|k| k.include? tag} #verifies if there are any bots with prefix 'mtf'
  return pipeline_hash #if it exists returns the hash with the content of pipeline.conf
else #else it adds the bots to the pipeline
  #add asn-lookup-bot to destination-queue of entrance_point
  if pipeline_hash[entry_point].has_key? 'destination-queues' #verifies if the key
"destination-queues" exists
  pipeline_hash[entry_point]['destination-queues'][ << tag + '-'+ asn_name + '-' +
queue_name #if it exists adds asn-bot
  else
    pipeline_hash[entry_point]['destination-queues'] = [tag + '-'+ asn_name + '-' +
queue_name] #adds key "destination-queues" and asn-bot-queue as the value
  end
```

Figure 5.16: Connection of ASN-expert to the Existing Pipeline

Then its created the source and destination queues for the *asn-expert-bot*, and it was added to the destination-queue the bot following the *asn-expert* in the pipeline, which is the *modify-expert*. This process, the generation of the source and destination queues is repeated for every single bot in the pipeline. The queues related to the target-entities are created dynamically and it is used their names and unique identifier to build their source and destination queues, as shown in figure 5.17.

All the generated hashes, with the source and destination queues for every bot are then saved into the ”main hash”, that was previously created by the reading of the *pipeline.conf* file at the beginning of this method. Finally, the queues inside of the ”main hash” are ordered alphabetically.

The next method of this section is equivalent to the writing method of the Runtime Generator. The method *write_pipeline* writes the output of the function *read_pipeline* into the file *pipeline.conf* and it is also generated backup files, one of the original configuration (before adding the new bots) and another with the new pipeline, both files are generated

with a timestamp in their names.

Finally, the method *clean_pipeline* removes the queues that were generated by the Ruby program, returning a clean file. This method searches for the given tag to the bots, in this case it was used the tag ”dcy”, and it deletes all the queues that contain this tag. Then is returned the new and clean pipeline which contains only the ”original” bots.

```
#add entities filter and output bots source and destination queues
ENTITIES.each do |entity|
  name = entity.name
  id = entity.classification_id

  queue_modify_entities['source-queue'] = tag + '-' + name + '-' + modify_name + '-'
+ queue_name
  queue_modify_entities['destination-queues'] = [tag + '-' + name + '-' + filter + '-' +
queue_name]
  modify_queue_aux = Marshal.load(Marshal.dump(queue_modify_entities))
  pipeline_hash[tag + '-' + name + '-' + modify_name] = modify_queue_aux
  queue_filter['source-queue'] = tag + '-' + name + '-' + filter + '-' + queue_name
  queue_filter['destination-queues'] = [tag + '-' + rabbit_mq_name + '-' +
queue_name]

  array_output.each do |bot|
    if bot.match(tag+'-'+id)
      queue_filter['destination-queues'] += [bot + '-' + queue_name]
      queue_filter_aux = Marshal.load(Marshal.dump(queue_filter))
      pipeline_hash[tag + '-' + name + '-' + filter] = queue_filter_aux
    end
  end
end
```

Figure 5.17: Entities Source and Destination Queues

5.7 Command Line Switch

For the execution of the *"intelmq_configurations_generator.rb"* it was necessary to create a Command Line Switch, which gives several options to execute the program via command line. The first part of the Command Line Switch is where the commands to execute the program are parsed. For the parsing it is used a Ruby Class named *GetoptLong*, it allows a POSIX (Portable Operating System Interface)-style options like *–file*, as well as single letter options like *-f*, as shown in figure 5.18.

```
opts = GetoptLong.new(
  [ '--help', '-h', GetoptLong::NO_ARGUMENT ],
  [ '--execute', '-e', GetoptLong::OPTIONAL_ARGUMENT ],
  [ '--asn', '-a', GetoptLong::OPTIONAL_ARGUMENT ],
  [ '--modify', '-m', GetoptLong::OPTIONAL_ARGUMENT ],
  [ '--runtime', '-r', GetoptLong::OPTIONAL_ARGUMENT ],
  [ '--pipeline', '-p', GetoptLong::OPTIONAL_ARGUMENT ],
  [ '--del_runtime', '-t', GetoptLong::REQUIRED_ARGUMENT ],
  [ '--del_pipeline', '-l', GetoptLong::REQUIRED_ARGUMENT ],
  [ '--res_pipeline', '-c', GetoptLong::OPTIONAL_ARGUMENT ]
)
```

Figure 5.18: Parsing of the Command Line Options

The following part is where all the methods are called to be executed, according to the given command. For example, by examining figure 5.19, is possible to verify that the command – *execute* or *-e* executes all the methods at once, meaning that all configurations files will be generated at the same time. Another example is, if the command *–asn* or *-a* is used, then only the file which contains the ASN database will be generated. By executing the methods separately is an easier way for debugging, in cases where a method fails to execute or to produce the expected output. The *–help* command prints to the command line all the existing options to execute correctly the Ruby program.

```
opts.each do |opt, arg|
  case opt
    when "--execute"
      write_asn_database(config_file)
      generate_modify_conf_file(arg,config_file)
      generate_modify_entities(config_file)
      generate_others_modify_file(config_file)
      generate_special_modify_url_fqdn(config_file)
      write_runtime(arg,config_file)
      write_pipeline(arg,config_file)
    when '--asn'
      write_asn_database(config_file)
```

Figure 5.19: Command Line Options

5.8 Outputs

The purpose of this section is to summarise the obtained outputs from the methods described earlier in this chapter. Figure 5.20 shows a schema of the outputs of each method.

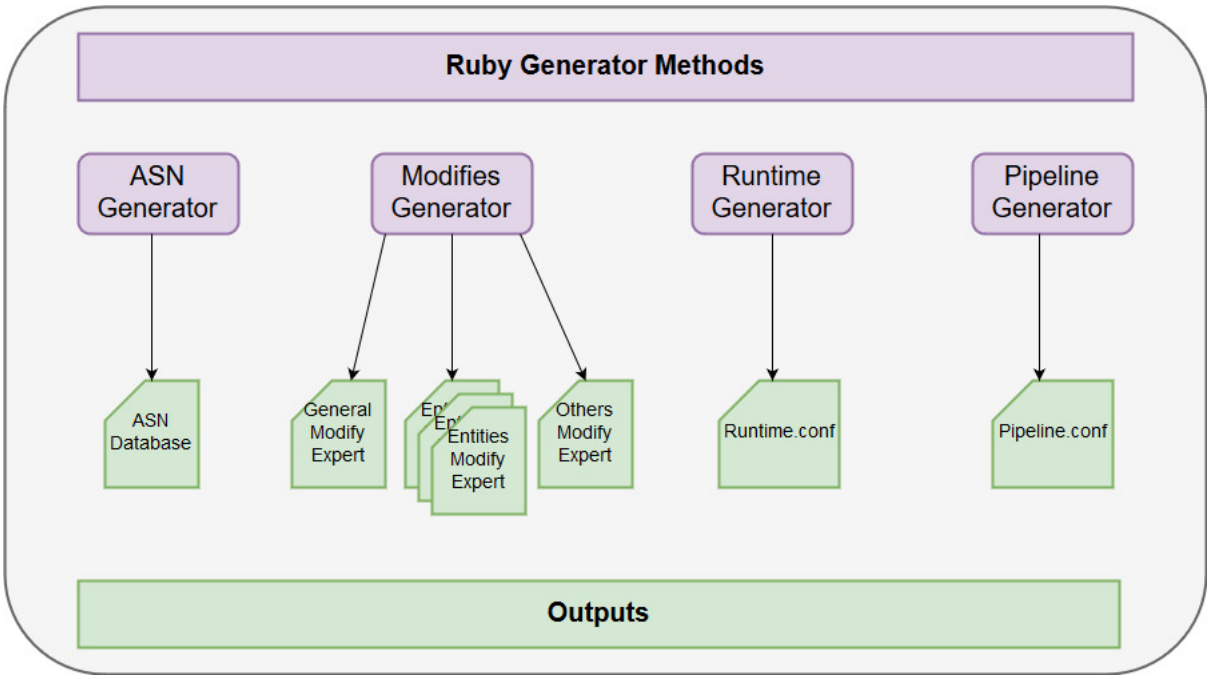

Figure 5.20: Generated Outputs

The files generated for this project were the following:

- ASN database
- General modify-expert configuration file
- Entities modifies-experts configuration files
- Others modify-expert configuration file
- Runtime.conf
- Pipeline.conf

By generating these files with the correct configurations, IntelMQ is ready and prepared to filter the events, according to the attributes of each target-entity.

5.9 Conclusion

This chapter described the implementation of the Ruby program, which generates all the configurations files necessary to filter Cyber Intelligence events, according to a set of attributes of the target-entities, through IntelMQ. This implementation was divided into

separate sections, and each one is responsible for the production of a set of configuration files. The first section explains the structure of the program as a whole, while the following sections describe in more detail every method and how they work. It is also explained how the program is executed and what commands are necessary to run it. The last section of this chapter shows a summary of all the configuration files that were generated by the program.

The next chapter shows the tests that were made to the program and the results obtained, to understand if the new pipeline and the corresponding configuration files can filter the events, according to the attributes of the target-entities.

Chapter 6

Results and Evaluation

After developing the Ruby program, which is responsible for the generation of configuration files and rule sets for the filtering of Cyber Intelligence events, according to the most relevant attributes of the target-entities, it was necessary to verify, evaluate and to understand if the events were being filtered correctly. This chapter presents the results obtained of the filtering of Cyber Intelligence events by IntelMQ using the configurations generated by the program *"intelmq_configurations_generator.rb"*. It is explained the test conditions, in other words, how the test was performed and then it is made an analysis of the obtained results.

6.1 Test Conditions

To perform the test to verify if the events were filtered correctly, IntelMQ was installed in a machine with Red Hat Enterprise Linux (RHEL) operating system, with 95GB of disk space, 20GB of RAM and 8 CPUs. IntelMQ processes millions of events, so it was necessary a robust machine to perform the test. IntelMQ was up and running between the 11^{th} of July to the 10^{th} of August, the test had the duration of 30 days, this range of days allows to have a better confidence on the final results.

The original IntelMQ pipeline (the pipeline that was already on IntelMQ) was composed by 108 *collector-bots*, 98 *parser-bots* and 15 *expert-bots*. The Cyber Intelligence Feeds from where the collector-bots retrieved the events were described on table 3.1 from the third chapter. After executing the *"intelmq_configurations_generator.rb"* it was added to the original pipeline 33 bots (*experts* and *outputs*) for the target-entities. In total the final pipeline had 254 bots, and its structure is shown in figure 6.1.

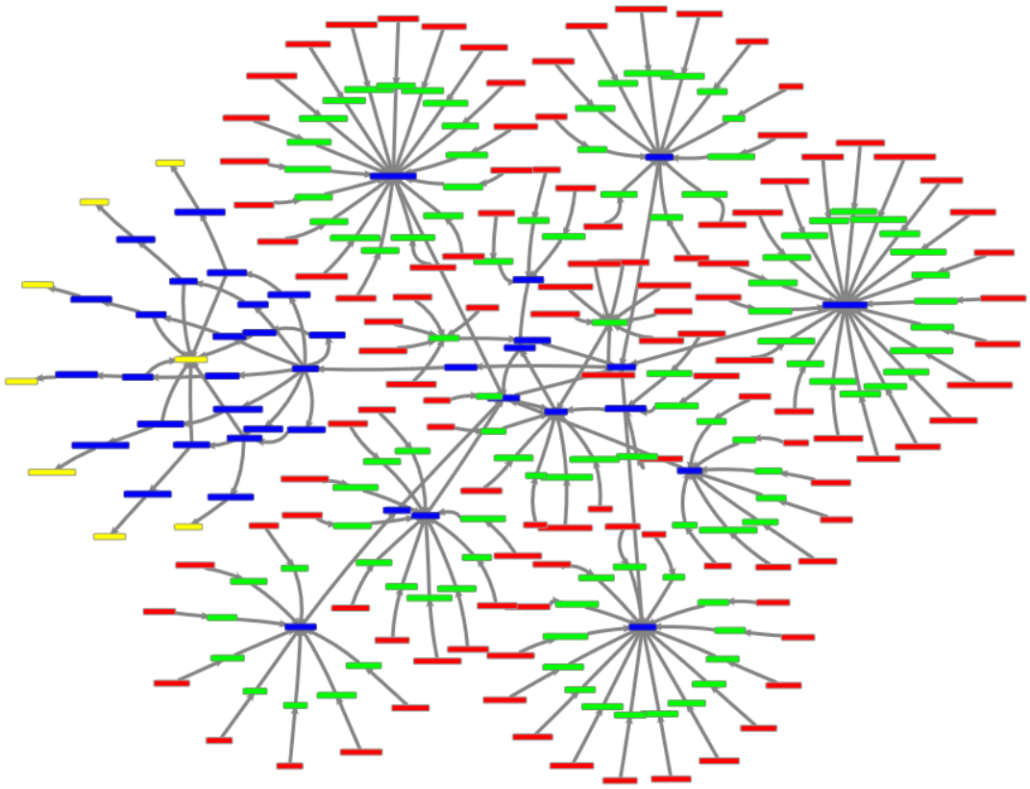

Figure 6.1: IntelMQ Final Pipeline

Figure 6.1 shows the final format of the IntelMQ pipeline, the pipeline generated by the Ruby script is on the left side of the image, and is composed by *expert-bots* (blue) and *output-bots* (yellow).

For the test it was used the entities A, B, C and D, with information that was found by Maltego, as described previously. However, it was created three more entities with IP addresses which belong to entities A, C and D. These IP addresses were retrieved by the Cyber Watch Program of DCY. These entities were named as entity-A1, entity-C1 and entity-D1, to differ from the others with information found by Maltego. Table 6.1 shows the total of IP addresses provided by the Cyber Watch Program, for each entity.

The test was performed with the total of 7 entities, with the duration of 30 days and with the purpose of filter events, according to the attributes of each target-entity (IP addresses, FQDNs and URLs).

Table 6.1: Entities Total Events

Entities	**Number of IP Addresses**
A1	78
C1	517
D1	23

6.2 Analysis of Results

This section describes the results obtained by the filtering of events. First, it is given an overall analysis of the filtering, then is made an analysis for each entity, and finally it is made an analysis of the type of events that were filtered by IntelMQ.

6.2.1 Global Analysis

As mentioned previously, IntelMQ was up and running for 30 days straight. In total, it filtered **64,816,823** events, where **330,525** belong to the target-entities and **64,486,298** are related to *others* events. Figure 6.2 shows a graph of the total of events filtered by IntelMQ during the 30 days.

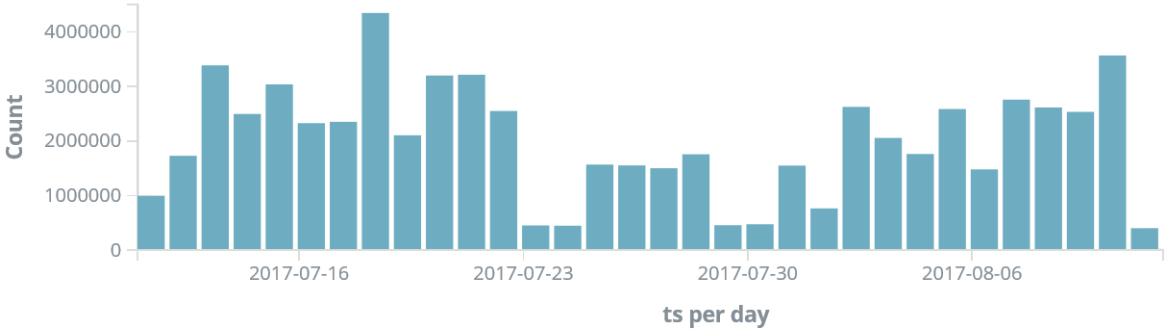

Figure 6.2: Daily Total Events

By observing figure 6.2 is possible to verify that, for most of the days, IntelMQ filtered around **1,000,000** of events, a part from the days between the 24th and 31st of July. This decrease in the filtering of events was due to adjustments made to the *collector-bots*. Every *collector-bot* retrieves information from the Cyber Intelligence Feeds in certain periods of time, for example, every hour the *collector-bot* fetches a report of events. However, most of the Cyber Intelligence Feeds are updated once a day, or once a week, meaning that if the *collector-bot* was configured to get the reports hourly, it would get repeated events, consuming resources and time unnecessarily. So, the *collector-bots* were reconfigured to retrieve the information, according to the time that the Cyber Intelligence Feed was updated.

To avoid an extreme resource consumption on the machine where IntelMQ was running, another adjustment was made, which was also responsible for the decreasing of events, in the mentioned period of time. The *collector-bots* had to fetch the reports intercalated, this way was possible to create a balance in the machine resources, allowing to IntelMQ to run without problems. For example, if the machine ran out of RAM, due to extreme resources consumption, i. e., *collector-bots* fetching the information at the same time, IntelMQ would stop running. The solution found to this problem, it was to turn off some *collector-bots*, for a small period of time (10 / 15 minutes), and turning them

back on at different times, this way they would get the reports intercalated, creating the necessary balance in the machine resources.

After the adjustments, the number of events filtered daily was very consistent, being between **1,000,000** to **2,000,000** of events. Some days have more events than others, because the size of the reports of the Cyber Intelligence Feeds vary according to the number of events reported on that day or week.

6.2.2 Global Entities Analysis

This subsection presents the global analysis of the filtered events on the target-entities.

Table 6.2 summarises the total of events filtered for each target-entity.

Table 6.2: Entities Total Events

Entities	Total Events
A	1,134
A1	318,912
B	0
C	0
C1	10,377
D	194
D1	0

By observing table 6.2 is possible to verify that the entity with the highest number of events is **entity-A1**, followed by **entity-C1** with the second highest number of events, then there are entities **A** and **D**. Entities **B**, **C** and **D1** have zero events.

One of the reasons why **entity-B** has none events, it might be because of the size of the organisation, it is a very small company and less likely to be a target of attacks.

The obtained metadata on **entity-C** by Maltego was a very small sample, when compared to the other entities (see table 4.1), this could be a reason of why **entity-C** has none events. Its attributes found by Maltego were not used as targets, or they were not reported on the used Cyber Intelligence Feeds. However, its IP addresses from Cyber Watch were involved in several events, as seen on table 6.2 (**entity-C1**).

Entity-D1 has zero events, because the IP addresses provided by Cyber Watch were not involved in malicious activities, or they were not reported on the used Cyber Intelligence Feeds. However, the metadata found by Maltego on **entity-D** was composed mainly by DNSs and URLs (see table 4.1) and most of the events filtered for this entity involved more network names than network addresses (this will be explained in more detail in subsection 6.2.5).

6.2.3 Entities A and A1

In this subsection it is presented the results obtained of the filtering of events during the 30-day test on entities A and A1, they represent the same entity, however entity-A has metadata provided by Maltego, while entity-A1 has IP addresses obtained by the Cyber Watch Program.

Entity-A

Figure 6.3 presents the graph of the daily filtering of events for entity-A, during 30 days.

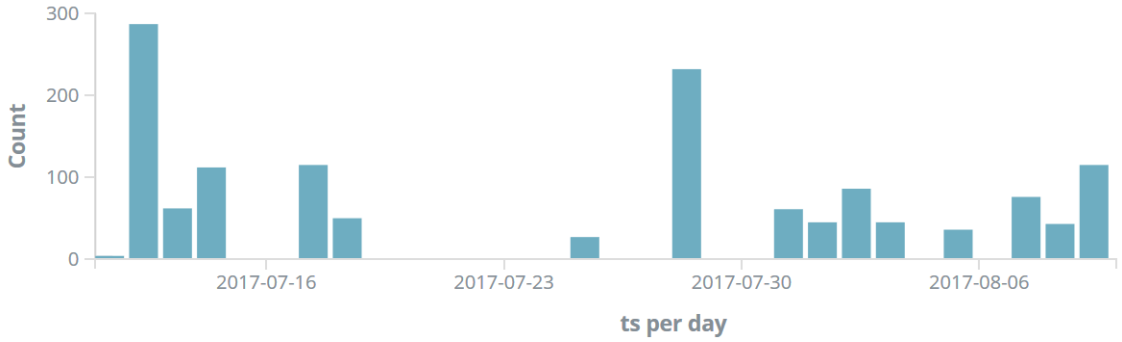

Figure 6.3: Daily Events of Entity-A

By observing figure 6.3 is possible to verify that the daily pattern of the filtering of events for entity-A matches with the daily total of events (figure 6.2). At the beginning of the test the number of filtered events was higher, then it decreased between 24th and 31st of July due to the adjustments made to the *collector-bots*, after this period the number of events increased.

The maximum of events reached in one day was on the 12th of July and the 28th of July, with the total of **286** and **208** events, respectively, while on the other days the average of events is around **65**. During the 30-day period there were 15 days with none events.

The type of events filtered for entity-A, involved only its IP addresses, there were not any events containing URLs and / or FQDNs belonging to this entity.

As mentioned previously, the total of events filtered for entity-A was **1,134**, where only **123** were unique. It is very common to have repeated events on the Cyber Intelligence feeds, because while they are not updated and / or the attack / vulnerability it is not solved, the same event will be reported several times. The Cyber Intelligence feed that reported **99%** of the events for entity-A it was *BitSight Ciberfeed Stream* corresponding to *malware* events, the remaining **1%** belongs to the *ShadowServer* feed which reported *vulnerable services*.

Entity-A1

The following graph shows the daily filtering of events for entity-A1.

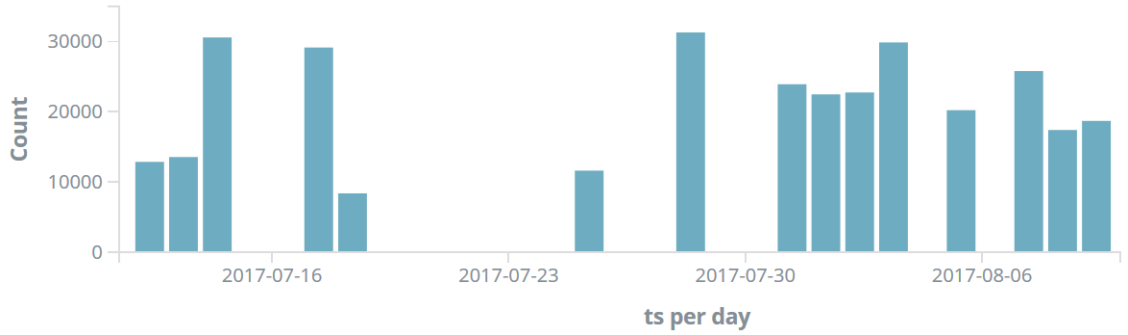

Figure 6.4: Daily Events of Entity-A1

The presented graph is similar to figure 6.3, however it contains a greater number of filtered events per day. The peak of events in one day for entity-A1 was on the 28th of July with the total of **27,446**. The average of events on the remaining days where exists filtered events is aproximately **20,813**. Similarly to entity-A, there were 14 days with zero events.

From the **318,912** events filtered for entity-A1, only **22** were unique events. As previously stated, the rules for the filtering of events for entity-A1 only had IP addresses, consequently, the filtered events for this entity, only involved its IP addresses. Once again, the Cyber Intelligence feed *BitSight Ciberfeed Stream* reported **99,9%** of the events, which are related to *malware*. The remaining **0,1%** were reported by the feed *ShadowServer*, where the events are related to *vulnerable services*.

Comparing both entities, A and A1, is interesting to verify that the obtained results were very similar, despite entity-A1 had a greater quantity of events than entity-A.

Both entities only had events involving IP addresses, and the events were reported by the same Cyber Intelligence feeds. The IP addresses found by Maltego and the ones provided by Cyber Watch were different, meaning that both configuration files and, consequently, the filtering of events for both entities, complemented each other, providing a greater knowledge on the type of events where these entities were involved.

6.2.4 Entity-C1

On this subsection is presented the analysis of the filtering of events for entity-C1. Figure 6.5 presents the 30-day graph of the filtered events.

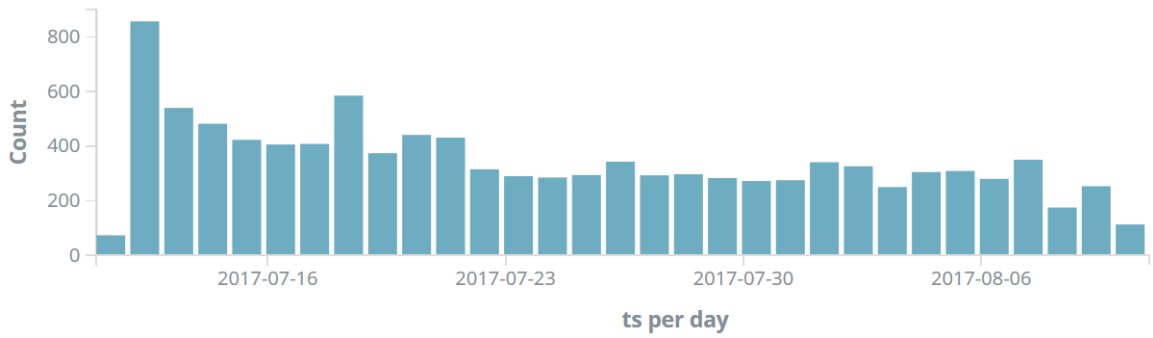

Figure 6.5: Daily Events of Entity-C1

By observing figure 6.5 is possible to see that the daily pattern of the filtering of events is very different from the previous entities. On the first week, there is higher number of filtered events, then it decreases but the filtering is very consistent throughout the remaining days. The day with the highest value of filtered events was the 11th of July with the total of **1225** events. The average of events filtered per day for this entity is around **345**. There were events filtered for entity-C1 every day from the 30-day period.

The filtered events of entity-C1 only involved IP addresses, and there are **716** unique events, in other words, unique IP addresses from the **10,377** events. The filtered events for entity-C1 were reported from several Cyber Intelligence feeds, table 6.3 summarises the total events reported by each feed for entity-C1, as well as the type of events that each feed reported.

Table 6.3: Entity-C1 Total Events By Each Cyber Intelligence Feed

Feed	Total Events	Percentage of Events	Type of Events
Abuse.ch Feodo IPs	57	0,55	Command & Control
AlienVault Reputation List	2547	24,5	Malware
AlienVault OTX	3	0,029	Blacklist
Bambenek C2 IPs	1	0,001	Command & Control
Blocklist.de	448	4,3	IDS Alert and Brute-Force
CI Army	5498	53	Blacklist
Danger Rulez	57	0,55	Brute-Force
DataPlane	830	8	Scanner and Brute-Force
NetLab360	117	1,12	Scanner
NoThink	29	0,28	DDoS
Taichung	237	2,28	Scanner and Malware
Turris Grey List	553	5,32	Scanner

Observing table 6.3 is possible to verify that the Cyber Intelligence feed, which reported the majority of events for entity-C1, it was *CI Army*. This feed provided events with black listed IP addresses, and represents **53%** of the total filtered events. The second largest percentage of events belongs to the feed *AlienVault Reputation List*, with **24,5%** and this feed reported events of the type *malware*.

Comparing the filtered events for entity-C1 with the other entities, the filtered events for entity-C1 are more diversified, the events were also reported from several feeds and the type of the events vary as well. The filtering of events for entity-C1 were more diverse and more consistent throughout the 30-day test than the others target-entities.

6.2.5 Entity-D

This subction presents the analysis of the filtered events for entity-D, figure 6.6 shows the graph of the daily filtering of events during the test period.

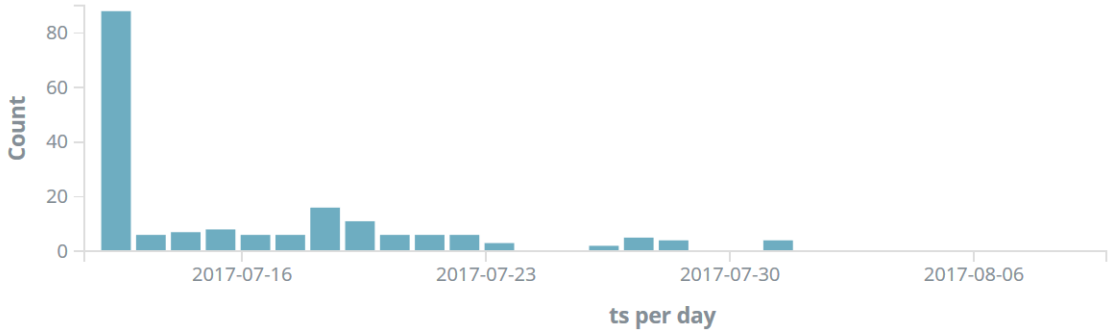

Figure 6.6: Daily Events of Entity-D

Similarly, to the previous entity, the day with the maximum of filtered events it was the 12th of July, with the total of **88** events. The first week of the test had more filtered events then the following weeks, as previously shown. The average of events filtered per day for this entity is around **6**. There were 13 days with none events, during the test for entity-D.

From the **194** events filtered for entity-D, **46** events are unique. These events involve IP addresses, URLs and FQDNs from entity-D, where some of events contain these three fields, while others only contain one of three fields. Table 6.4 shows the quantity of events which contain the three fields or only one of them.

Table 6.4: Total Events per Field

Fields in the Event	Total Events
IP Address, URL, FQDN	30
IP Address	86
FQDN	78

The majority of the events filtered for entity-D contain IP addresses, meaning that most of them were filtered taking into account this field, while the events that only contain the field FQDN, were filtered according to the rules for the FQDNs of the target-entity.

The filtered events for entity-D were provided by three different Cyber Intelligence feeds. Table 6.5 presents the total of events reported by each Cyber Intelligence feed and the type of events that each feed reported.

Table 6.5: Entity-D Total Events By Each Cyber Intelligence Feed

Feed	Total Events	Percentage of Events	Type of Events
BitSight Ciberfeed Stream	6	3,1	Malware
HP Hosts	78	40,2	Blacklist
ShadowServer	110	56,7	Malware

By observing table 6.5 is possible to verify that most of the events were reported by *ShadowServer*, representing **56,7** of the reported events, the type of events that this feed provided were related to malware. *HP Hosts* is the second feed with the largest amount of reported events for entity-D, this feed provided events with black listed domains. Finally, there is the feed *BitSight Ciberfeed Stream* representing **3,1** of the reported events and providing information on events related to malware.

The events reported for entity-D are slightly different from the previous entities. For this entity there are several events containing URLs and FQDNs, while the events on the others target-entities are exclusively related to IP addresses. When comparing the configuration files of all target-entities, entity-D had the largest number of FQDNs and URLs, this might explain why this entity has events related to public network names.

6.3 Conclusion

The functioning of the filtering of events is measured through the obtained results showed on this chapter. However, the results also depend on the metadata, which served as input for the configurations for the filtering of the events. This means that, if there is rule for a certain IP address / FQDN or URL and during the 30-day test there were no events containing this network address / network name, it does not mean that the filtering did not work, it means that the attribute of the target-entity was not involved in any malicious activities.

There were seven target-entities for the 30-day test, and as shown on this chapter, there were filtered events for four of the seven entities. From the entities scouted and mapped by Maltego 2 out of 4 had filtered events. And the three entities with IP addresses provided by Cyber Watch, 2 out of 3 had filtered events.

In cases where an attribute of a target-entity suffers an attack, or it is vulnerable or it is involved in malicious activities, the event is filtered correctly, and it will be sent to the respective file as well as to Hidra, being then possible to forensically analyse each event. It is possible to conclude that the filtering of events does work, if the attribute of a target-entity is in an event.

Chapter 7

Conclusion and Future Work

Taking into account that business value, data and personal information are increasing and rapidly migrating into a digital form on open and interconnected technology platforms, the risks from cyberattacks become very intimidating. There are several ”enemies” that an organisation should be aware of, from criminals that seek for financial gains through fraud and identity theft, to rivals / competitors that steal intellectual property or disrupt business to be in an advatange, as well as *hacktivists* which try to pass through online firewalls to make political statements.

Most companies have difficulties to manage their capabilities in cyber risk, they lack on information to make powerful decisions, and the traditional strategies of ”securing the perimeter” are being insufficient. Most organisations also struggle on quantifying the impact of risks and mitigation plans. Much of the damage results from an inadequate response to a breach rather than the breach itself.

This is why that at Portugal Telecom there are several security operations, where each of them is responsible for different security areas. One of the strategies that PT adopted, in order to have a resilient cyber protection is to analyse and understand the type of attacks that an organisation was exposed to. Here is where IntelMQ comes in, this platform has the capability to collect, deduplicate, classificate, tag and filter Cyber Intelligence events where a certain organisation was involved.

During this project it was developed a Ruby program capable of configuring IntelMQ to filter events, according to certain attributes of four different target-entities (organisations). However, for testing purposes, it was created seven target-entities, for three of these four entities, it was created extra input data for IntelMQ, provided by the Cyber Watch operation of Portugal Telecom, making the total of seven target-entities. All four organisations were scouted and mapped by Maltego, in order to find their public information on the Internet. This way, it was possible to find their attributes that were more likely to suffer an attack, since they are exposed on the Internet.

On the 30-day test accomplished on this project, there were filtered events for four out of the seven target-entities. As the main goal of this project was the implementation of an

architecture which filters Cyber Intelligence events, according to certain characteristics of an organisation, it can be said that this goal was met. Despite three entities had no events. This does not mean that the filtering did not work for these entities with zero events. If there were events with one or more attributes of an entity, the event would be filtered for the corresponding entity. The reason why there were three entities with zero filtered events, it was because there were no events containing their attributes, which in one hand is a good sign. It means that these entities did not suffer an attack, or they do not have vulnerable services or were not involved in malicious activities.

The filtering of Cyber Intelligence events allowed to identify each event, according to the respective entity, the events after being identified and filtered can be sent to several analysis platforms allowing a forensic analysis to the event, in order to better understand what, how and who triggered the event and perfomed the attack.

There is always room for improvements, and this project is no exception to the rule. There are several aspects that should be taken into account for future work and improve the filtering of events. Regarding the Ruby program developed for this project, the code itself should be more dynamic. At the moment, if for some reason it is necessary, for example, to add another *expert-bot* to the pipeline that the program generates, it will be required to update the code, in order for the file *pipeline.conf* to have the correct connections between the bots.

To reduce the number of duplicated events, it should be added to the pipeline generated by *"intelmq_configurations_generator.rb"*, *deduplicator-bots* after the *filter-bots* for each target-entity, this way the number of repeated events would decrease substantially.

Finally, it should be used different tools to scout and map the target-entities, beyond Maltego. Thus, would be possible to compare results between the different tools, and create more complete YAML files with the most relevant attributes on the Internet of the target-entities. In addition, the obtained metadata from Maltego should be converted into the YAML format automatically, instead of being created manually as happened in this project.

Abbreviations

APIs Application Programming Interface. 21

AS Autonomous System. 13

ASN Autonomous System Number. 13

CCI Cyber Counter Intelligence. 9

CERT Computer Emergency Response Team. 23

CIDR Classless Inter-Domain Routing. 13

CINS Cargo Incident Notification System. 29

CnC Command and Control. 28

CPUs Central Processing Unit. 73

CSD CyberSecurity Development and Integration. 20

CSIRT Computer Security Incident Response Team. 23

CTAS Commercial Transform Application Server. 22

CVE Common Vulnerabilities and Exposures. 14

DCY Direção de Cyber Security and Privacy. 1

DGA Domain Generated Algorithms. 28

DNSs Domain Name Server. 19

DoD Department of Defense. 7

DoS Denial of Service. 28

FCUL Faculdade de Ciências da Universidade de Lisboa. 2

FQDNs Fully Qualified Domain Name. 4

FTP File Transfer Protocol. 28

GB Gigabytes. 73

HIDRA High performance Infrastructure for Data Research and Analysis. 20

HTTP Hypertext Transfer Protocol. 14

HUMINT Human Intelligence. 8

ICM IntelMQ Configuration Manager. 5

ID Identifier. 30

IDS Intrusion Detection System. 9

IMAP Internet Message Access Protocol. 28

IOC Indicators of Compromise. 12

IP Internet Protocol. 4

IRC Internet Relay Chat. 28

IT Information Technology. 23

iTDS internal Transform Distribution Server. 22

JSON JavaScript Object Notation. 26

MI Mestrado em Informática. 2

MIME Multipurpose Internet Mail Extensions. 14

MX Mail Exchanger. 37

NS Name Server. 37

OSINT Open Source Intelligence. 8

OTX Open Threat Exchange. 28

PEI Projeto em Engenharia Informática. 2

PID Process Identifier. 30

PT Portugal Telecom. 1

RAM Random Access Memory. 73

RHEL Red Hat Enterprise Linux. 73

SIEM Security Information and Event Management. 13

SIGHUP Signal Hang Up. 31

SIGINT Signals Intelligence. 31

SIP Session Initiation Protocol. 28

SNMP Simple Network Management Protocol. 29

SOCMINT Social Media Intelligence. 8

SSH Secure Shell. 28

TLD Top Level Domain. 30

U.S. United States. 7

URIs Uniform Resource Identifier. 14

URLs Uniform Resource Locator. 4

Bibliography

[1] James Britt and Neurogami. Marshal. https://ruby-doc.org/core-2.2.2/Marshal.html, February 2017.

[2] CERT. Silk. https://tools.netsa.cert.org/silk/, November 2016.

[3] Eric Chabrow. Cyber intelligence: What exactly is it? http://www.bankinfosecurity.com/blogs/cyber-intelligence-what-exactly-it-p-1061, September 2011.

[4] Threat Connect. 7 threat intelligence tools your cybersecurity team needs. https://www.threatconnect.com/blog/7-threat-intelligence-tools-your-cybersecurity-team-needs/, December 2016.

[5] Department Of Defense. Joint publication 2-0. https://fas.org/irp/doddir/dod/jp2_0.pdf, October 2013.

[6] Alison Deighton. Cyber security: the dos, the don'ts and the legal issues you need to understand. https://www.financierworldwide.com/cyber-security-the-dos-the-donts-and-the-legal-issues-you-need-to-u#.WYHY2TIp_CK, November 2015.

[7] Docker. What is docker? https://www.docker.com/what-docker, November 2016.

[8] Bank Of England. Cbest intelligence-led testing. http://www.bankofengland.co.uk/financialstability/fsc/Documents/anintroductiontocbest.pdf, June 2016.

[9] ENISA. Incident handling automation. https://www.enisa.europa.eu/topics/csirt-cert-services/community-projects/incident-handling-automation, November 2016.

[10] Clark C. Evans. Yaml: Yaml ain't markup language. http://yaml.org/, December 2016.

[11] EY. Cyber threat intelligence - how to get ahead of cyber-crime. http://www.ey.com/Publication/vwLUAssets/EY-cyber-threat-intelligence-how-to-get-ahead-of-cybercrime/$FILE/EY-cyber-threat-intelligence-how-to-get-ahead-of-cybercrime.pdf, November 2014.

[12] FireEye. Fireeye isight intelligence. https://www.fireeye.com/products/cyber-threat-intelligence.html, December 2016.

[13] Nick Furneaux. Maltego machines and other stuff. http://www.csitech.co.uk/featured/, April 2013.

[14] Gartner. Threat intelligence: What is it, and how can it protect you from today's advanced cyber-attacks? http://www.bankinfosecurity.com/blogs/cyber-intelligence-what-exactly-it-p-1061, August 2014.

[15] IT Governance. Cyber security risk assessments (10 steps to cyber security). http://www.itgovernance.co.uk/cyber-security-risk-assessments-10-steps-to-cyber-security.aspx, December 2016.

[16] G. Huston. Autonomous system (as) number reservation for documentation use. https://tools.ietf.org/html/rfc5398, December 2008.

[17] IntelMQ. Data harmonization. https://github.com/certtools/intelmq/blob/master/docs/Data-Harmonization.md, November 2016.

[18] Json. Introducing json. http://www.json.org/, July 2017.

[19] Joshua Lawton. In search of the mythical 400 pound hacker. https://www.linkedin.com/pulse/search-mythical-400-pound-hacker-joshua-lawton-mba-pmp-csm/, September 2016.

[20] Nick Lee. Business data at high-risk through employee use of cloud storage. http://www.questcloudcomputing.co.uk/business-data-at-high-risk-through-employee-use-of-cloud-storage/, November 2015.

[21] Robert Lee. Cyber counterintelligence: From theory to practice. https://www.tripwire.com/state-of-security/security-data-protection/

cyber-counterintelligence-from-theory-to-practice/, May 2014.

[22] Robert Lee. Cyber intelligence collection operations. https://www.tripwire.com/state-of-security/security-data-protection/cyber-intelligence-collection-operations/, February 2014.

[23] Robert Lee. Cyber threat intelligence. https://www.tripwire.com/state-of-security/security-data-protection/cyber-threat-intelligence/, October 2014.

[24] Robert Lee. An introduction to cyber intelligence. https://www.tripwire.com/state-of-security/security-data-protection/introduction-cyber-intelligence/, January 2014.

[25] Cricket Liu. Strengthen your network security with passive dns. http://www.infoworld.com/article/2994016/network-security/strengthen-your-network-security-with-passive-dns.html, October 2015.

[26] Tiago Mendo. Monitoring pastebin. https://mendo.pt/monitoring-pastebin/, March 2015.

[27] Springer Nature. Cyber threat intelligence feeds. http://www.nature.com/webfeeds/index.html?foxtrotcallback=true, March 2017.

[28] Angela Nichols. The best threat intelligence feeds. https://www.anomali.com/blog/the-best-threat-intelligence-feeds, September 2016.

[29] NSA. Signals intelligence. https://www.nsa.gov/what-we-do/signals-intelligence/, May 2016.

[30] Jim Orrill. What is the difference between active and passive vulnerability scanners? http://smallbusiness.chron.com/difference-between-active-passive-vulnerability-scanners-34805.html, July 2017.

[31] PORTUGUESE PARLIAMENT. Act on the protection of personal data. https://www.cnpd.pt/english/bin/legislation/Law6798EN.HTM, October 1998.

[32] Paterva. Maltego transforms. https://www.paterva.com/web7/docs/M3GuideTransforms.pdf, January 2011.

[33] Paterva. What is maltego? https://www.paterva.com/web7/buy/maltego-clients/maltego.php, November 2016.

[34] Charles Pham. From events to incidents. https://www.sans.org/reading-room/whitepapers/incident/events-incidents-646, November 2001.

[35] Jason Polancich. Cyber intelligence: Defining what you know. http://www.darkreading.com/operations/cyber-intelligence-defining-what-you-know/a/d-id/1319257, February 2015.

[36] RabbitMQ. What is rabbitmq? https://www.rabbitmq.com/, November 2016.

[37] Redis. redis. http://redis.io/, November 2016.

[38] Margaret Rouse. hacktivism. http://searchsecurity.techtarget.com/definition/hacktivism, June 2007.

[39] ShadowServer. Get reports on your network. https://www.shadowserver.org/wiki/pmwiki.php/Involve/GetReportsOnYourNetwork, July 2017.

[40] SiteTakeDown. Brand abuse explained. https://sitetakedown.com/brand-protection/brand-abuse-explained/, March 2017.

[41] Solarwinds. What is netflow? http://www.solarwinds.com/what-is-netflow, November 2016.

[42] John Reed Stark. Cyber-security due diligence: a new imperative. https://www.complianceweek.com/blogs/john-reed-stark/cyber-security-due-diligence-a-new-imperative#.WYHbczIp_CJ, June 2016.

[43] Portugal Telecom. Csirt. https://conteudos.telecom.pt/Documents/EN/pt/security/portugal-telecom-csirt.pdf, February 2015.

[44] The Cyber Threat. What is a web feed? http://thecyberthreat.com/cyber-threat-intelligence-feeds/, March 2017.

[45] TutorialsPoint. Penetration testing - manual an automated. https://www.tutorialspoint.com/penetration_testing/penetration_testing_manual_automated.htm, July 2017.

[46] TutorialsPoint. Ruby - classes and objects. https://tools.ietf.org/html/rfc5398, July 2017.

[47] John Vanderzyden. Welcome to the elk stack: Elasticsearch, logstash, and kibana. https://qbox.io/blog/welcome-to-the-elk-stack-elasticsearch-logstash-kibana, July 2015.

[48] Webroot. Computer security threats. https://www.webroot.com/us/en/home/resources/articles/pc-security/computer-security-threats, July 2017.